NYANTO

2024 SDD MAJOR PROJECT DOCUMENTATION



TIMOTHY YANG

Table of contents

Check 1 **5**
- Problem statement: 5
- Initial Design specifications 6
  - Developer specifications 6
  - User specifications 6
- Gantt Chart of entire project (predicted) 7
- Screen designs 8
- Storyboard 10
- Context Diagram 10
- Discussion of selection of language to be used 11
- Social / Ethical considerations 12
  - Copyright/Plagiarism 12
  - Accessibility/inclusivity 13
  - Cultural sensitivity and children-friendly design 13
  - Privacy 13
  - Ergonomics 13

Part A **14**
- Five modules and functions with a brief description 14
  - Modules 14
  - Functions 15
- Pseudocode and system (algorithm) flowchart of ONE module 16
  - Pseudocode 16
  - Flowchart 17
- IPO Chart of the main module 18
- Structure Chart of ALL modules 19
- Data Flow Diagram 20
- Files types used with their respective Data Structures 20
- Data Dictionary 22
- Platform/OS considerations & hardware implications 25

Check 2 **26**
- User Interface items selected 26
  - Sliders 26
  - Checkbox 26
  - Command Buttons 27
  - Icon-only buttons 27
  - Input prompts (labels)/command keys 28
- EBNF and Railroad diagrams for C# If and Switch statement 28
  - EBNF vers 28
  - Railroad diagram vers 29
- Test data 31
  - Code tested (irrelevant minor things like playing a sound effect and changing colour of cats have been omitted for clarity sake since they are not being tested): 31

The test data 34
Error checking 42
Stubs 42
Flags 44
Debugging output statements 45
Desk check of one module 48
Breakpoint, program traces and single line stepping 49
Breakpoints 49
Program Traces 50
Single Line stepping 51
Readability of code 53
Following naming conventions 53
Commenting 54
Formatting 54
Discussion of syntax errors, runtime errors, logic errors 56
Syntax 56
Logic 56
Runtime 58
Part B **59**
Efficiency and elegancy of code 59
Optimisation of performance 59
Adaptability to future input changes 60
Modularity 61
Reducing clutter in module mainline 61
Refine the user interface Final UI design (screenshot) 62
Main Menu 62
Game Mode Selection Menu 62
Settings Menu 63
Tutorial 63
Credits Page 64
Screen in a Game Mode 64
Pause Menu 65
Game Over 65
CASE tools to monitor changes and version control 66
Catering for possible changing user requirements 67
User manual 68
About Nyanto: 68
Getting started (after following the installation manual) 68
Settings 68
Game modes 68
How to play 69
How to win 69
Installation manual (Game ONLY, not source files) 69
Digital copy of all source code 70

Project report 71
What I’ve learnt 71
Future directions 72
Comparison with original specifications 73
Logbook 76
Bibliography 84

Check 1

Problem statement:

A sufficiently challenging project must be created to demonstrate and develop my algorithmic skills and understanding of the principles for software design as specified by the NSW syllabus by the middle of 2024. The project should also not require extensive time to develop or learn the tools to create it (feasible to have done bulk within 6 weeks) as this will be created within my HSC year. Creating a casual standalone Windows PC game that is easy to pick up in Unity and involves some intellectually challenging elements and luck as well should be suitable as a solution as I am already acquainted with the basics of Unity. Furthermore, this can serve as a demonstration of coding experience when applying for jobs in the future.

For the genre of the game, I have chosen to make a Suika-like due to its current popularity and simplicity in concept. Suika-likes are a genre of game that exploded in popularity in late 2023 after the success of the “Suika Game” on the Nintendo Switch by Aladdin X. The “Suika Game” however was not where the game concept of merging 2 identical objects in a container to prevent the container overflowing originated. The Chinese browser game 合成大西瓜 was the first to come up with this concept and it was inspired by the merging mechanic of 2 of the same object from 2048. In terms of performance issues for creating this game there should be none due to its simplicity. There also shouldn’t be any compatibility issues with existing software or hardware since the game does not rely on any third-party applications and requires no extra or specific hardware outside what would be found on a modern PC (since Windows 10 release) since it shouldn’t be computationally expensive.

Games of the Suika-like genre on PC typically only have one game mode, lack support for keyboard play, and have an emphasis on competition through point score rather than novelty fun. Often times too, the colliders of the combining items do not align with the sprites for them causing frustration where items that are visually touching are not combined. Therefore to have a Suika-like game available without these plaguing issues that often cause player frustration I will offer an open-source solution that aims to showcase resolutions to these issues and promote further innovation in the game genre while retaining the original essence of a Suika game. The boundary and scope for this project are limited to the above requirements for creating an offline open-source game that any modern Windows PC user can play.



^collage of popular games in the Suika-like genre.

This project will be submitted as part of coursework for the SDD course.

Initial Design specifications

Developer specifications

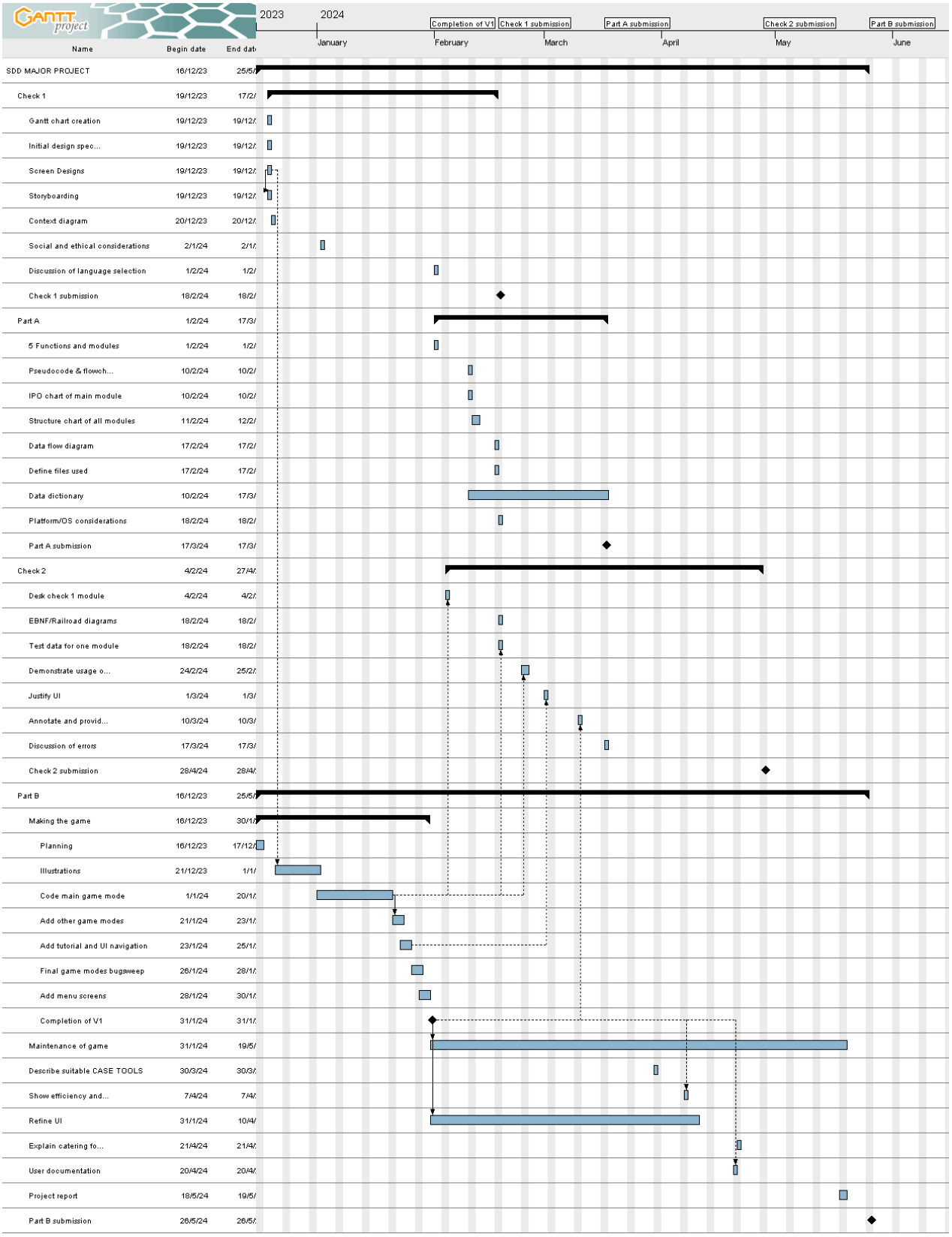
- The overall development will loosely follow a structured approach and switch to an agile approach to account for any minute changes until the submission deadline
- Utilise copyright-free audio files for sound effects and background music and tune using Audacity
- Create illustrations required for the game using Adobe Illustrator
- Game development conducted using the Unity game engine and C#
- Code formatting is consistent with C# naming conventions such as Pascal for functions/classes and camel case for variables
- Design code for reusability and modularity
- Code is internally documented with comments to assist other developers in understanding the logic as an open-source project
- Code generates logs of debugging statements for major modules
- The entire system should be mapped out with a structure chart
- Handle version control with Git & GitHub
- Select play testers from a variety of people to test quality throughout development with WebGL builds and survey for errors/ergonomic issues
- Design algorithm architecture to accommodate platform porting with Unity Input action system

User specifications

- UI is navigatable by mouse and keyboard in an ergonomic way and intuitive
- Buttons of the next page are defaulted when using keyboard to navigate the UI
- UI and screen design use standardised icons and has consistent design (including font) that is intuitive
- Appropriate and non-intrusive audio feedback for game events and UI usage
- Colour of UI elements used complement each other and cause no visibility issues
- Game has clear and simple ingame tutorial
- Compatibility with modern computers that have Windows 10/11 installed
- Resolution of screen suitable for 16:10 and 16:9 aspect ratios
- Appropriate messages or prompts (neutral tone) given for navigation or notable game events
- Game can be paused with keyboard shortcuts “p” or “esc”
- Respects user privacy and security and asks if necessary for data collection
- Game files are below 2GB in size and require no special hardware
- While running the game should not take any more than 1gb of RAM
- Game does not require an internet connection
- Smooth running when on at least 30fps with no freezing, long response times or crashing issues

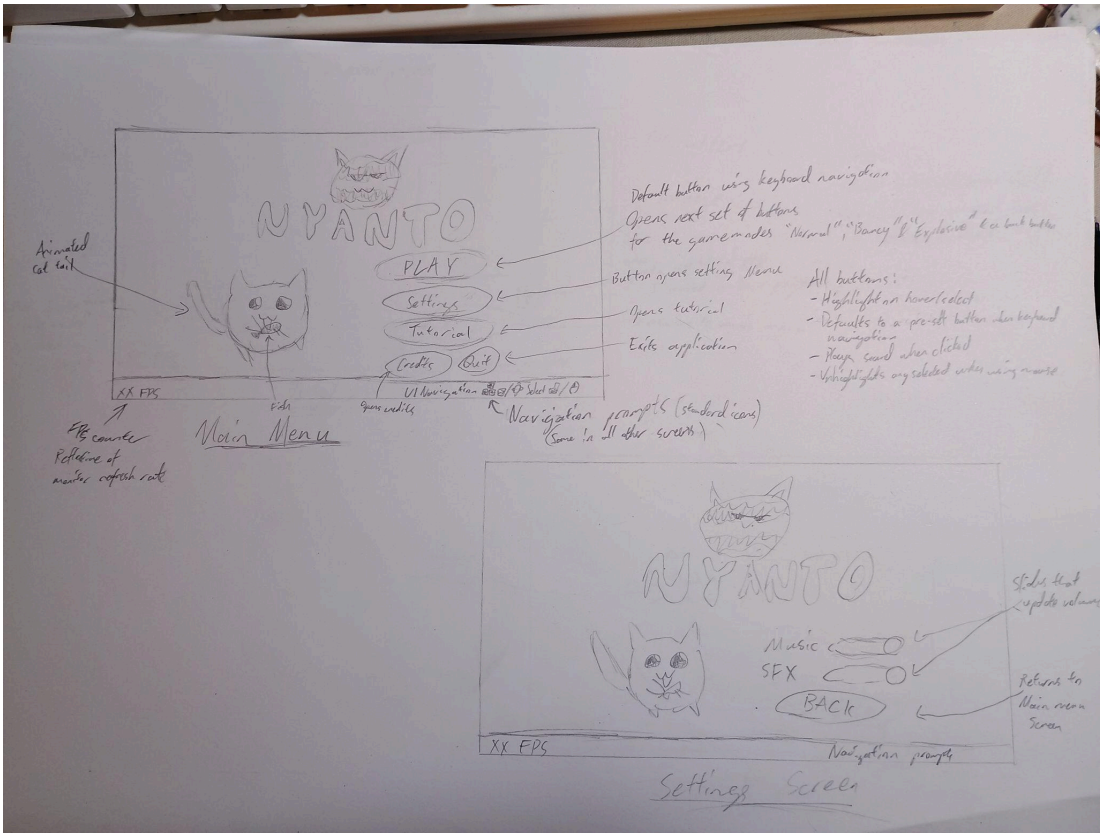
Gantt Chart of entire project (predicted)

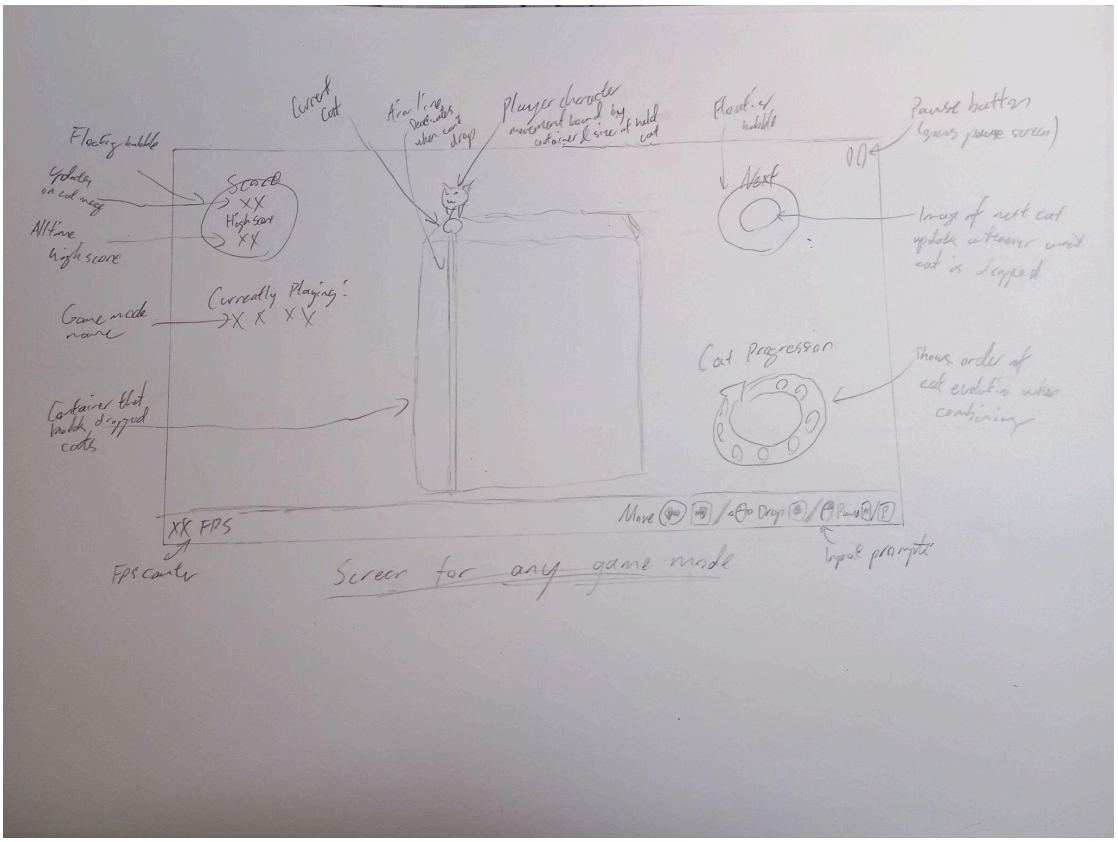
For better resolution please download or view the image from the source files on GitHub: (https://github.com/TYProjectUploader/Nyanto/tree/main/Major%20project%20theory%20images)

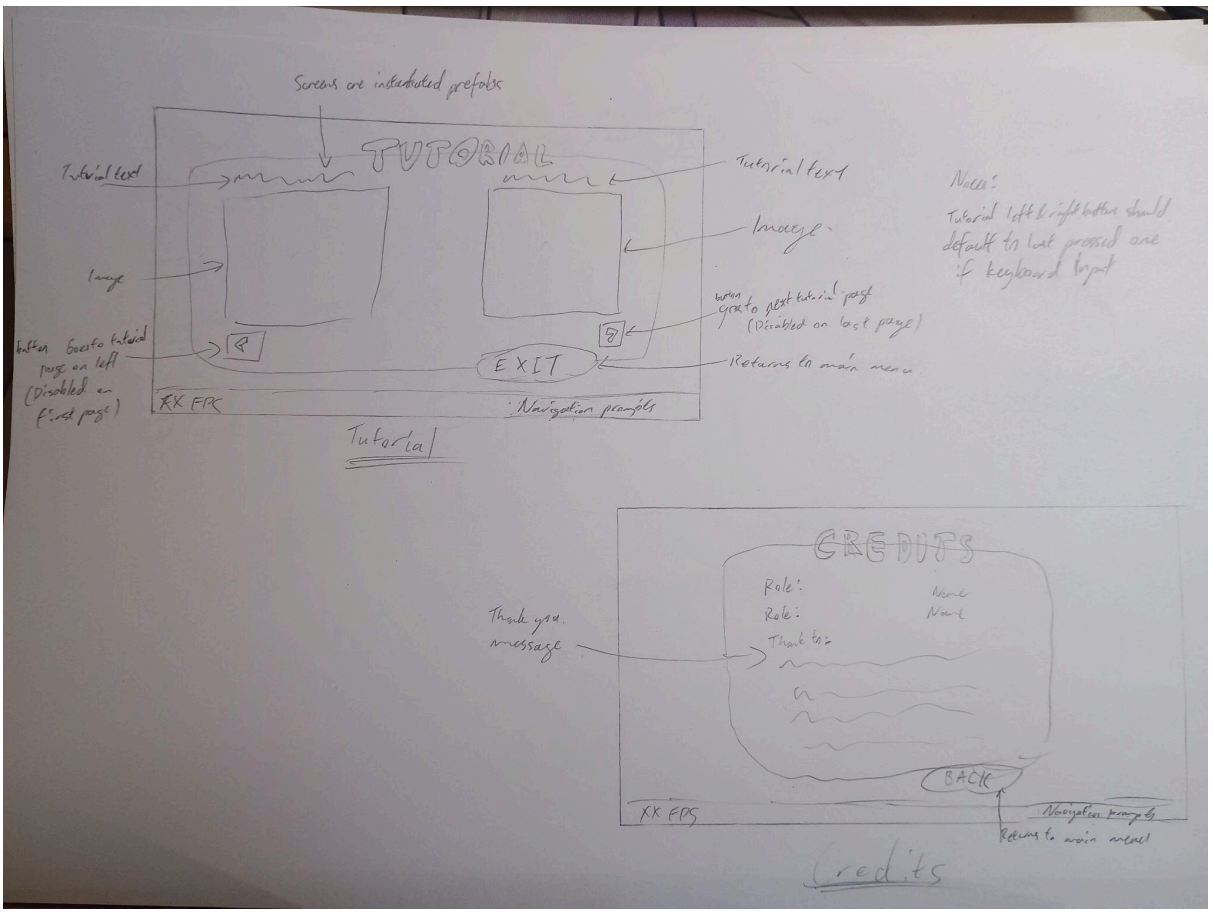


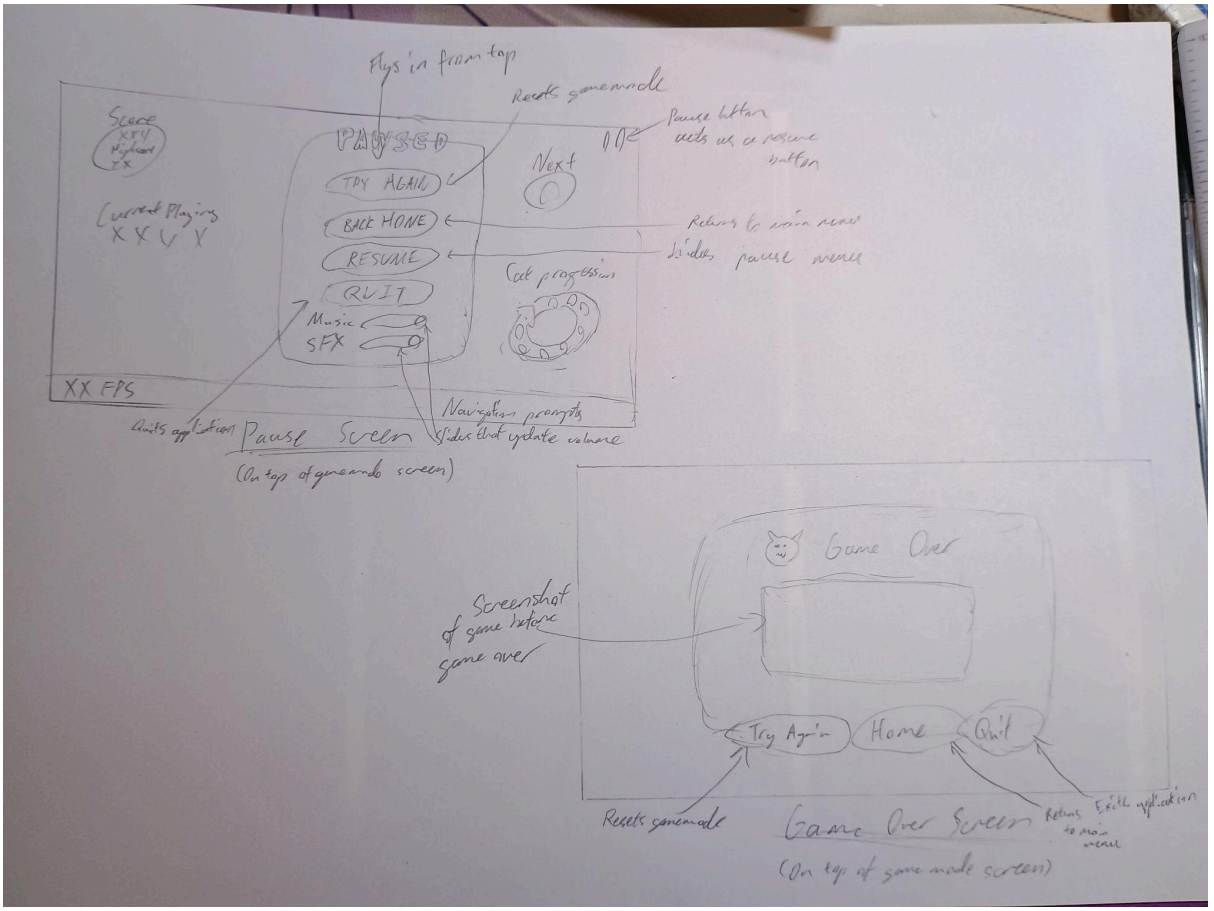
Screen designs

For better resolution please download or view images from the source files on GitHub: (https://github.com/TYProjectUploader/Nyanto/tree/main/Major%20project%20theory%20images/Screen%20designs)



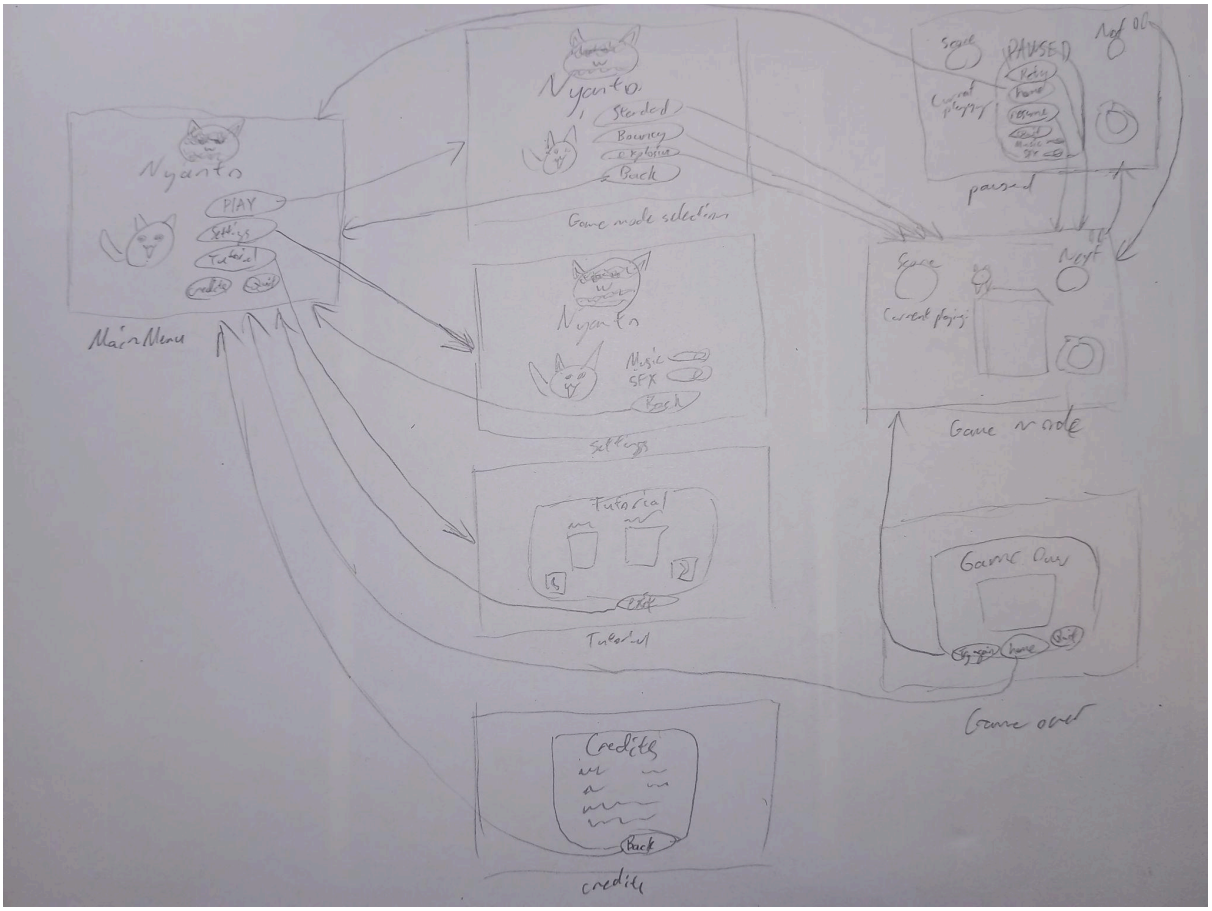



Screens are instantiated prefabs
TUTORIAL
Tutorial text
Tutorial text
Image
Image
Button goes to tutorial page on left (Disabled on first page)
Button goes to next tutorial page (Disabled on last page)
EXIT
Returns to main menu
XX FPS
Navigation prompts
Tutorial
Notes:
Tutorial left & right button should default to last pressed one if keyboard input
CREDITS
Role:
Name
Role:
Name
Thank to:
Thank you message
BACK
XX FPS
Navigation prompts
Returns to main menu
Credits

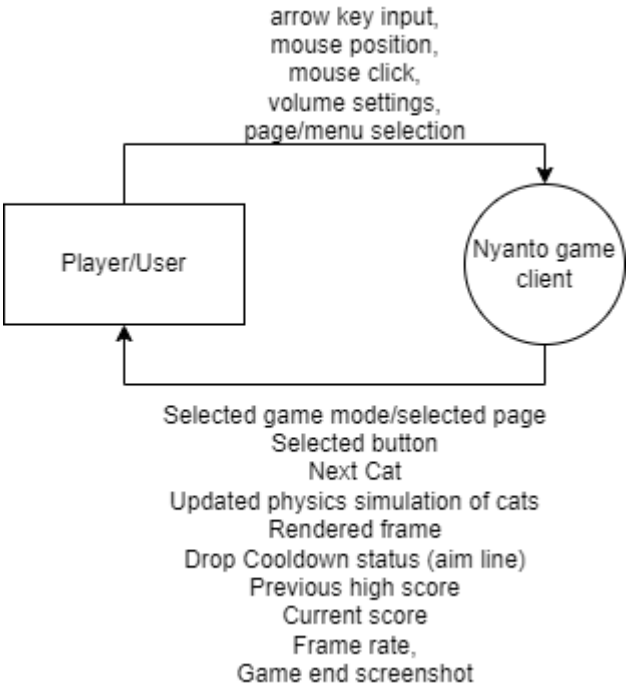

Flys in from top
Resets game mode
Score
XXII
Highest
XX
PAUSED
Pause button acts as a resume button
TRY AGAIN
Next
BACK HOME
Returns to main menu
Hides pause menu
RESUME
QUIT
Cat progression
Current Playing
X X V Y
Music
SFX
XX FPS
Quits application
Navigation prompts
Slides that update volume
Pause Screen
(On top of game mode screen)
Game Over
Screenshot of game before game over
Try Again
Home
Quit
Resets game mode
Game Over Screen
Exits application
Returns to main menu
(On top of game mode screen)

Storyboard

For better resolution please download the image from the source files on GitHub:
(https://github.com/TYProjectUploader/Nyanto/tree/main/Major%20project%20theory%20images)



Context Diagram



Discussion of selection of language to be used

C# is the chosen language for this game project due to having past experience with it. In addition to being the primary language for scripting in Unity which is a widely adopted game engine, it has a robust framework, library collection, and community support due to its popularity in software development. A benefit of this popularity is that the text editor Visual Studio Code which I will be using for this project also has extensions for C# and Unity IntelliSense. C# also has syntax similar to other popular languages such as Java and C++, allowing skills learnt over this project to transfer to future projects outside of C#. All of this culminates to allow for all features of the solution to be feasibly developed.

Some of the main built-in benefits of using C# for development are its automatic garbage collection which frees up memory that is no longer used by the application, seamless cross-platform development with the .NET framework that compiles code into the Common Intermediate language (CIL), and its design as an object-oriented programming (OOP) language. Being an OOP it is perfect for my project where the logic is driven by the user, the modularity allows for code to be easily managed, maintained, and added to as changes to one part of the system don't affect others with major benefits of this being the ease of debugging and collaboration. Code can also be easily reused as classes can be created based on existing ones, inheriting attributes and behaviours to minimise redundancy and promote a more efficient development process. It should also be noted that having the code compiled into CIL allows for Just-in-Time compilation which is what enables cross-platform development and dynamic code loading by converting the CIL to machine code native as the computer runs the program.

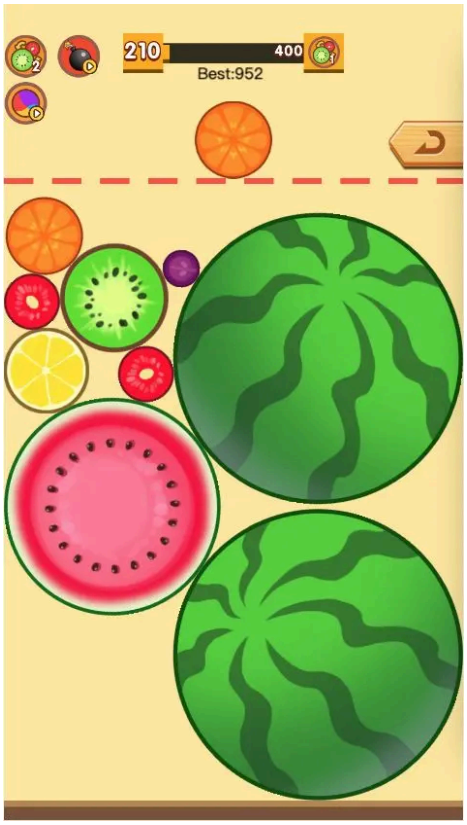
The main drawbacks in choosing C# is the inability to manage memory manually and other system-level operations since it is automatic as well as its lower performance compared to languages directly compiled into native machine code such as C++. Although the benefit of automatic garbage collection is the lack of need to manage memory manually, it can cause noticeable performance degradation temporarily (freezing or low frames) when it occurs every so often especially in games. Also, a small nuisance of C# being a compiled language is its requirement of code being recompiled when changes are made before it can be run.

Social / Ethical considerations

Copyright/Plagiarism

Copyright refers to the legal right given to creators over the publishing, distribution, and use of their work. This is an ethical consideration that not only must be made with my work but also with my use of other's work. With the Unity game I created, it was not feasible to create all the assets for the game myself with high quality, especially the music. Therefore, assets imported from the internet must have licence agreements that specify the ability for free use. This is to ensure my use of other people's work was not violating the creator's terms of use. Furthermore, all models, art, and music not belonging will be acknowledged in the bibliography below and in a text file attached to the final version when distributed on GitHub with links to where they were acquired even if the author says it's not required.

The game is based on the Suika Game available on Nintendo Switch (developed by AladdinX) which is by far the most popular version of the myriad of games (WhaleGameOnline, HololiveSuika, and My Suika just to name a few) inspired by Meadow Science's (米兜科技) web browser game called "合成大西瓜". To avoid infringing on copyrights of Meadow Science's original game and the many others that made similar versions of their game, the design will not be like the many knock-offs that copy every fruit and gameplay mechanic. My main differentiation from those copies and Meadow Science's game include but are not limited to: my merging logic that involves pushing nearby cats (fruits) away, dropping mechanic, score calculation, addition of the combo mechanic, 2 different game modes, and self-made original art.



^Image of 合成大西瓜 gameplay

Regarding my rights given by copyright as the game creator, the game's distribution will be made open source using the Apache license 2.0 so that this project can be used to benefit others. All source files and assets will be made downloadable from GitHub, except for the ones I didn't create myself.

Accessibility/inclusivity

An accessibility consideration to be made is to have the game be easily accessible to most modern PCs running Windows. To achieve such the game will be easily downloadable from online and free to play with an installation guide. Controls will also be made so that players can play the entire game singlehandedly with mouse only, keyboard only, or both as per their preference. Another consideration made was for the visually or auditory impaired. Features implemented to help these users include: larger buttons/text, different tier cats having different sizes rather than just colours to distinguish them and having the music and SFX of the game be non-essential to the game's playability. Although this game will be limited accessibility by its platform availability, by choosing Windows computers as the target platform for the game the accessibility is maximised.
A tutorial has also been created to assist users not familiar with the Suika-like game genre.

Cultural sensitivity and children-friendly design

Since the game should be able to be enjoyed by anyone the game should not heavily represent any cultures and be appropriate for a global audience. Violence and anything related such as blood or weapons shall not be depicted. Designs and game concepts were made simple to accommodate children.

Privacy

As an offline game, there is no requirement for the storage or use of any personal user data. Therefore, there is no need to consider the compromisation of data concerning the game. The only thing that will be stored by the game would be the all-time high score and audio volumes set by the player in previous play sessions.

Ergonomics

As a basic casual game, there should be little loading time for the game to launch and start being playable so feedback for loading has been omitted from the user's view. The user interface design will have consistent elements for font and buttons with little variation and utilise universally recognised icons for the controls and user prompts. Visual feedback to increase intuitiveness has been considered through the drop line being reactivated when the player can drop and buttons being highlighted when hovered/selected.
In terms of navigation features, the game can be fully navigated using only one hand with keyboard only, mouse only, or switching between the two as per user preference. Further, as the game is likely to cause rage-inducing moments for the user, the retry button has been purposely placed at the top of the pause menu for quick access and the quit application screen has been included there too.

Part A

Five modules and functions with a brief description

Modules

(did before the clarification that they are the same as functions… so it has been left in. Modules have been defined here as how the C# community generally defines one; where it is a collection or namespace of functions/classes that carry out an overall task)

Name	Description
PlayerMovement	Module containing the continuously running functions while the game is not over for updating the player position based on user input as well as the ability to drop cats.
GameManager	The module that stores and runs the functions to be run such as disabling physics when the game over is triggered or resetting the time scale of the physics simulation back to 1 when the game is restarted.
OpeningMenuUINav	A module for the main menu that specifically handles how the buttons and pages are navigated so that it can be done seamlessly with either mouse or keyboard.
CombinationManager	This module contains the functions and logic for detecting when 2 identical cats touch and the processes that follow that. Some of the processes after 2 identical cats touch involve merging the cats, increasing the score, and handling the combo mechanic.
CatGenerator	Module that holds the functions for spawning a cat and adding it to the physics simulation in addition to generating the next cat after the user drops the current one.

Functions

Name	Module	Description
UpdateMovementBoundary	PlayerMovement	The radius of the currently held cat is input to the function when a new cat is placed in the player's hands. The variables for the max and min x positions of the player is updated to account for the width of the cat. Afterwards the player's x position is clamped as per values of the max and min x positions of the player.
FadeOutGame	GameManager	This function is called after game end is triggered. First it captures a screenshot of the current screen then changes the input prompts to the navigation controls for menus. This is followed by the process of darkening the screen over 1.5 seconds by decreasing the alpha value of a black panel over the game screen. Finally it activates the game over menu selection.
SetDefaultBtn	OpeningMenuUINav	A page is sent given to the function and it determines whether the page is active. If it is and the player is using a keyboard to navigate the UI, it sets the current selected button to a pre-assigned button on the active page if on it. For the tutorial specifically, the default buttons are chosen based on whether the player last pressed back or forward.
CombineCats	CombinationManager	This function is inputted the two identical cats to be combined. The two cat's have their colours flash white temporarily before they are both destroyed. A cat of the next tier is then spawned in the averaged positions clamped within the boundaries of the container. Cats nearby the new cats are slightly pushed away upon spawn to provide space for the new cat in the container.
SpawnCat	CatGenerator	The function takes the current cat or fish held in the next bubble and spawns it in the player's hand. Afterwards using a random number generator, either a fish or random cat is generated to replace the image of the previous cat/fish in the next bubble.

Pseudocode and system (algorithm) flowchart of ONE module

CombineCats is a module that takes 2 cats that are to be combined then destroys both and spawns in a cat of the next tier from an array of cat prefabs named “Cats”.

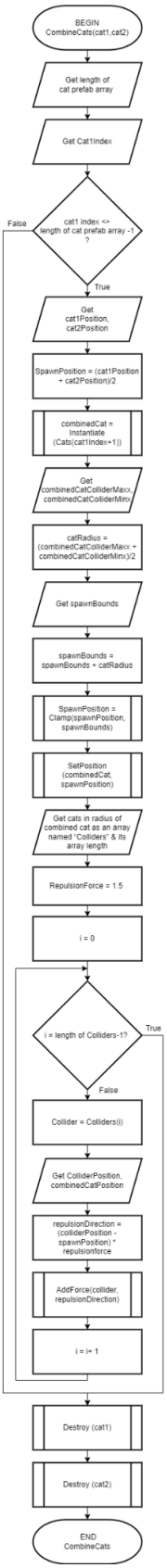
Pseudocode

Subprograms have not been defined as outside of scope the module

```
BEGIN CombineCats(cat1,cat2)
        Get length of cat prefab array
        Get cat1Index
        IF cat1Index <> length of catPrefabArray - 1 THEN
                Get cat1Position, cat2Position
                Let spawnPosition = (cat1Position + cat2Position)/2
                Let combinedCat = Instantiate(Cats(cat1Index+1))
                Get combinedCatColliderMaxx, combinedCatColliderMinx
                catRadius = (combinedCatColliderMaxx + combinedCatColliderMinx)/2
                Get spawnBounds
                spawnBounds = spawnBounds + catRadius
                spawnPosition = Clamp(spawnPosition, spawnBounds)
                SetPosition(combinedCat, spawnPosition)
                Get cats in radius of combined cat as an array named “Colliders”
                Let repulsionForce = 1.5
                FOR i=1 TO length of Colliders STEP 1
                        IF collider has a RigidBody2D
                                Collider = Colliders(i)
                                Get ColliderPosition, combinedCatPosition
                                repulsionDirection= (colliderPosition - spawnPosition)*repulsionForce
                                AddForce(collider, repulsionDirection)
                        END IF
                NEXT i
        ENDIF
        Destroy(cat1)
        Destroy(cat2)
END CombineCats
```

Flowchart

For better resolution please download or view the image from the source files on GitHub: (https://github.com/TYProjectUploader/Nyanto/tree/main/Major%20project%20theory%20images)

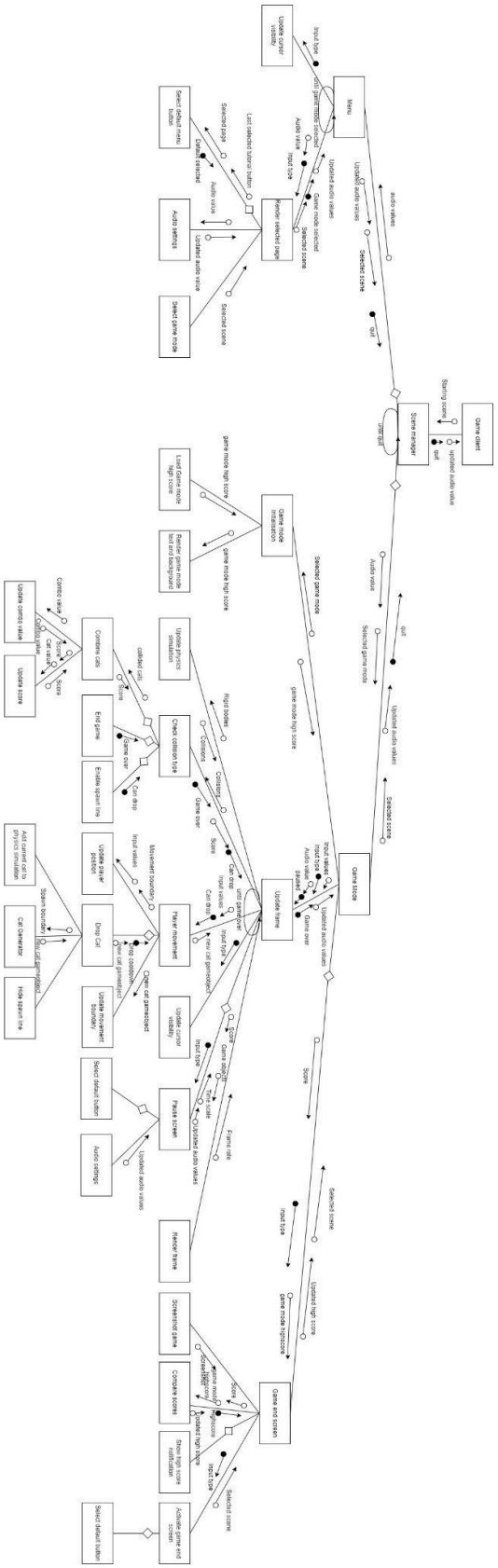


IPO Chart of the main module

Input	Process	Output
Left & right keyboard arrow keys/Mouse position	Determine if input type is keyboard or cursor Convert mouse position to screen coordinates Update player position based on input values	Hide cursor if non-cursor input is used or enable if cursor is used Game screen with new position of the player
Down arrow key/right mouse click	Add the current cat player is holding to physics simulation Disable ability to drop another cat Start cooldown to renable dropping Hide aimline Spawn the current cat in the “next” bubble in the player’s hand Generate a random cat image in the “next” bubble Update player movement boundary based on new cat	Current helding cat is dropped Updated game screen with next cat image updated for next image
Left click on button	Play button click sound Process relevant outputs for the button	Relevant menu/process to the button
AudioSliderValue	Update the stored audio values Change audio levels of the game client	Audio level of game client updated

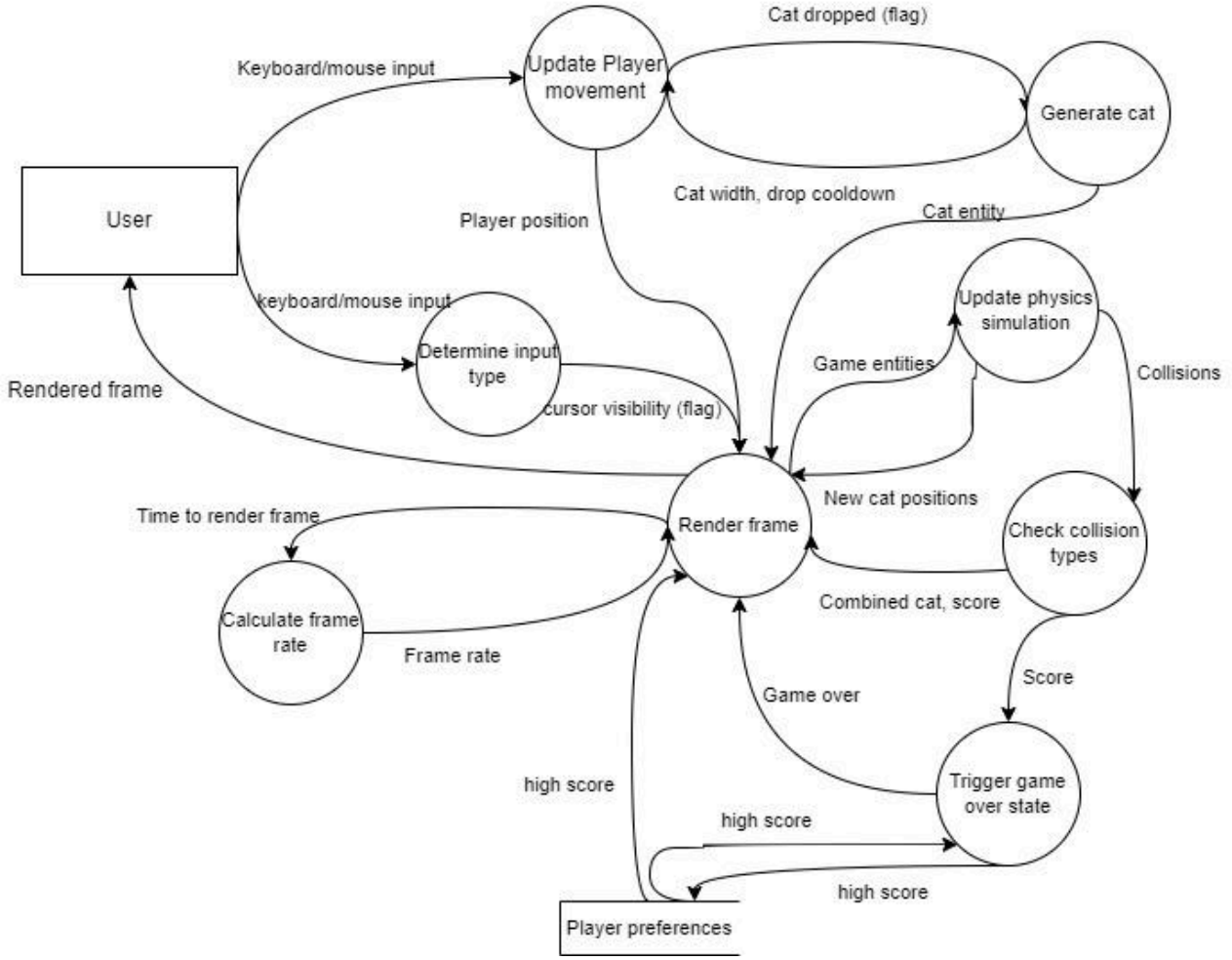
Structure Chart of ALL modules

For better resolution please download or view the image from the source files on GitHub:
(https://github.com/TYProjectUploader/Nyanto/tree/main/Major%20project%20theory%20images)



Data Flow Diagram

Data flow diagram for updating a frame when inside a game mode:



Files types used with their respective Data Structures

Only the file types that will be created/used in the Windows build have been listed below rather than the raw files used when creating the game in the Unity editor since Unity converts them to .assets, .dll or other file formats depending on the target platform for compilation.

File Type	File type & data structure applicable (sequential/relative access/arrays of records/multidimensional arrays)	File purpose/usage
.exe	Binary serialised data that is sequential but using jump functions can jump specific addresses or locations within the executable using branch instructions or function calls. (specific to windows only)	For Nyanto the exe file acts as an entry point with a sequence of instructions to running the Nyanto game application on the windows operating system. The exe file then evokes DLLs as required to run the functions for the game.

.dll	Data structures within a DLL depend on the specific algorithms and operations implemented within it. Nyanto will have DLL files with all data structures as it contains arrays (relative), code (sequential), and various project settings generated by Unity that would be stored in arrays of records and multidimensional arrays.	DLL stands for dynamic link library. DLL files are a type of file that contains compiled code resources and data that multiple programs can use simultaneously. Their main use is to allow for code reuse, efficient memory usage, and reduced storage space usage with modular components. The DLLs are read and called upon by the Nyanto executable as required at runtime to access files stored within them. All the C# source files used in the logic of Nyanto as an example are compiled into a DLL file named: Assembly-CSharp
.assets	Binary serialised data (sequential)	Various game assets are stored within any files labeled with .assets. Images, audio, models, materials and so on. These asset files often contain references to .resS files. The .asset files are generated by Unity during the compilation process using game asset files in their raw form (png, mp3 etc) supplied to the Unity editor. The serialised data allows quick access and use of required asset data when running the game.
.resS	A unity specific file type that uses binary serialised data (sequential)	Texture and mesh objects for the game are stored within these files by Unity to render objects with the GPU. This file type has little to no documentation online as only Unity knows how they are handled.
.json	Text file. This file type can be relatively accessed due to its formatting (with external tools) but this is a sequential file that can act as an array of records.	Json files are formatted in a specific way to store and exchange properties and attributes of objects. They are typically written to be readable by humans. In the Nyanto windows build it simply stores file names of the DLLs in an array and attributes of various objects to be initialised at the start of running the program.
.resources	Binary serialized data (sequential)	There is only one of these files in the build. Game assets and other data can be stored within this file and accessed here during runtime by Unity. The data stored within here is specified by source code or stored in the “resources” folder in the Unity Editor while building. This folder was not utilised in the final version of Nyanto but is during testing in the editor for storing the final screenshot.

.log	Text file (sequential)	There is only one of these text files that is generated and written into to include debugging output statements, logs generated by stubs and to store player data from previous executions of the program. This file is stored locally on the user's computer under appdata/locallow by default.
.info	Text file (sequential)	Only one of these text files exist and it is used to store information about the program. (Only used to store app name for Nyanto)
.config	Text file (sequential)	There are a few of these text file used to store configuration data for a few settings when running Nyanto.

Data Dictionary

Less than or 4 duplicates of the same type of data type has been included as per teacher instructions otherwise we'd be here for too long. Due to the game being built for 64-bit processors, all numbers other than double floating points (not used in this project) are stored at 4 bytes regardless of number size with floats specifically using IEEE standards. Strings are also stored using UTF-16 so byte storage size are 2*number of characters (due to using only english characters) in the string with no limit for length.

Data item	Data type	Format	no. bytes for storage	Size for display	Description	example	validation
Scene name	string		2 * Number of Characters	Number of Characters	A unique name assigned to each Unity Scene	ExplosiveMode	Name must not already exist
Tag	string		2 * Number of Characters	Number of Characters	A label used to group GameObjects	Cat	Tag must not already exist
persistentDataPath	string	C:\Users\<user>\AppData\LocalLow\<company name>	2 * Number of Characters	Number of Characters	The file path of files generated by the game stored on user's PC.	C:\Users\weeab\AppData\LocalLow\Nyanto	

GameObjectName	string		2 * Number of Characters	Number of Characters	Name label given to a game object class	Tutorial	
catIndex	integer	NNN	4	3	Index of the cat in its evolution	5	Integer between 0 and 999
comboCount	integer	NNN	4	3	The current value of combinations chained together	23	Integer between 0 and 999
combineValue	integer	NNNN	4	4	The point value for a given cat combine	6	Integer between 0 and 1000
Score	integer	NNNNNNNNNN	4	10	The current score of the player.	123440	Integer between 0 and 2,147,483,647 inclusive
Cat width	floating point	N.NNNNNNN	4	9	The distance between the smallest x and largest x point of a cat's collider	0.1239832	Must be greater than 0
Drop cooldown	floating point	N.N	4	2	Number of seconds before the player can drop again	0.9	Must be greater than 0.
timeSinceLastUpdate	floating point	N.NNNNNNN	4	9	The time elapsed since the Unity update function was last called	0.0021234	Must be positive number
volume	floating point	X.XXX	4	5	Value of the slider controlling audio volume.	0.324	Must be value between 0 and 1 inclusive

Paused	boolean	X	1	1	Whether game is paused or not. T or F	T	
hasCollided	boolean	X	1	1	Whether the cat game object has collided with something before. T or F	T	
GameOver	boolean	X	1	1	Whether game has ended. T or F	F	
Cat Images	Array of image files				Array containing all the sprites of the various cats	cat1.png cat2.png cat3.png	Images must be of png format
Cat prefabs	Array of gameobject classes				Array containing all the cat prefabs	Cat 1.prefab Cat 2.prefab Cat 3.prefab	
GameScreenshot	File (sequential)				The screenshot taken of the game when game over is triggered	gameOverScreenShot.png	Image must be of png format
CatSprite	File (sequential)				The sprite image of a cat	Cat1Sprite.png	Image must be of png or jpeg format
Player preferences	File (sequential)				Text file containing records of previously saved user settings		

Platform/OS considerations & hardware implications

Due to the time constraints on the project and taking into consideration my current mastery of coding, it would be best suited to design the game for only PC and specifically for Windows. Windows was selected as the supported OS for the game due to its widespread adoption on most PCs although at any time someone could recompile the game for Mac or Linux by downloading the open source files off GitHub. This would mean controls will be required to accommodate both mouse and keyboard usage. Great focus will need to be placed on keyboard interaction with the game as the widespread usage of laptops means mouse only control is not ideal due to the difficulty of piloting a trackpad to interact with the game.

With many PCs all having different display sizes, the UI and game elements have been designed to be able to resize and suit the two most commonly used aspect ratios of 16:9 and 16:10 without issues rather than being fixed pixel sizes.

As a simple game, the processing requirements and storage requirements won't be great so little consideration has been given to that. It should be able to run without any issues with any modern computer (past 5 years) that has a GPU since graphics are to be rendered. This however means that the game should be optimised in its physics calculations with tactics such as simplifying colliders and the collision calculations required.
These are the minimum specifications that I intend the game to be able to run smoothly on:

OS: Windows 10 or newer, 64-bit
Processor: Intel Core i3-6100 / AMD FX-8350
GPU: Intel HD graphics 530
Memory: 1 GB RAM
Storage: 1 GB

However, to allow the game to be quickly ported over to another platform if ever required, the controls of the game will be created using Unity's input action system. The input action system doesn't require a rewrite of game logic but only additions of bindings to control maps to allow interaction with various input devices such as touchscreen devices or joysticks.

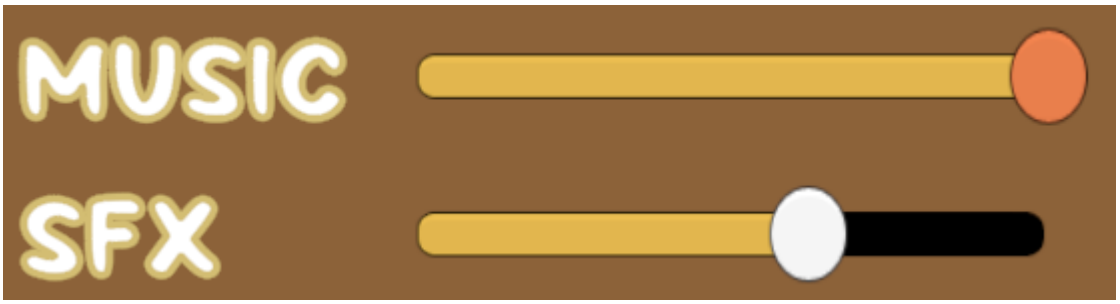
Check 2

User Interface items selected

Sliders

Sliders have been used for setting the values of the music and sfx volumes of the games. When they are selected the handle is highlighted orange to indicate it is being selected and the value can be changed by dragging the mouse or using the left/right arrow keys.

The use of sliders allows for precise volume inputs rather than discrete values such as increments of 0.1 up to 1(max). Furthermore sliders are a standardised UI design for volume interaction across many different software and platforms making this usage intuitive for the user.



Checkbox

A checkbox has only been used for turning screen shake on & off as it's the only setting that is a binary selection and doesn't require at least one option to be selected like the radio button. Future additions of binary selections to the settings can also be easily implemented in the same settings page. Furthermore, the checkboxes can be navigated and selected by both keyboard and mouse easily with a highlight of orange indicating when it's selected.

(selected)



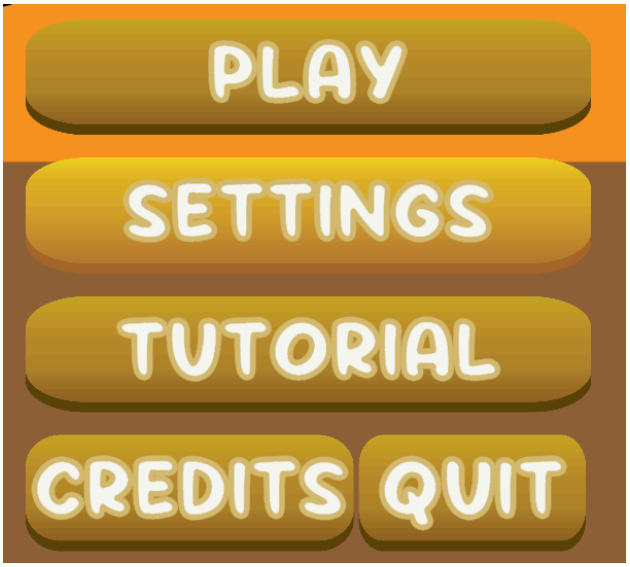
(unselected)



Command Buttons

Buttons have been extensively used throughout the GUI design as they are intuitive and straightforward for users to initiate actions/open menus for which the button has been labelled. These buttons have also been set up to be navigated by both mouse and keyboard with highlights to indicate when they are being selected, with the potential to be mobile-friendly if ever ported. Buttons are also easily customisable in size, shape and colour to complement the rest of the UI.

(Main menu buttons with settings being highlighted)



Icon-only buttons

These buttons feature only an icon to indicate their functionality at a glance without needing textual labels. They are very efficient in space usage, reducing the clutter in UI and can be widely understood and recognised across various cultures and languages due to their consistent use in software and digital interfaces. This is why they have been chosen for the pause button and next/back buttons on the tutorial.

(pause button)



(Tutorial next and back buttons)



Input prompts (labels)/command keys

These prompts provide clear and concise instructions required to perform actions within the game and are provided at the bottom of the screen based on which screen the user is currently on. They are placed to ensure that the user can easily see and reference them as needed yet unintrusive which allows them to skip straight to the gameplay if they ignore reading the tutorial. The pause menu has a command key that allows for the game to be interacted with keyboard only and to feel intuitive to users assimilated to playing other games as “esc” or “p” are both standard command keys to pause or escape from a game.

(UI navigation prompts)



(Gameplay input prompts & command key prompt)



EBNF and Railroad diagrams for C# If and Switch statement

“Statement” in the EBNF and railroad diagram below refers to any declaration statement, expression statement, jump statement, iteration statement, try-catch statement, and the if statement itself valid in the c# language. Statement and value have not been given an EBNF and railroad diagram, as that’d end up with me doing the entire language.

2 EBNF and railroad diagrams have been done because “Select a more complex statement if possible” was given as feedback during check 1.

EBNF vers

The if statement:

IfStatement = if“(“<condition> {(“||” | &&) <condition>}“){”{<statement>}“}”{<ElseIf Statement>}[else“{”{<statement>}“}”]

ElseIfStatement = else if“(“<condition> {(“||” | &&) <condition>}“){”{<statement>}“}”

Note: 2nd value has to be of same type and only numerical values can have a comparator using !=, >=, <=, > and <.

Condition = ([!]<boolean>) | (<value><comparator><value>)

Comparator = “==” | !“=” | >“=” | <“=” | > | <

value = <string> | <integer> | <float> | <boolean> | <object> | <function with a return value>

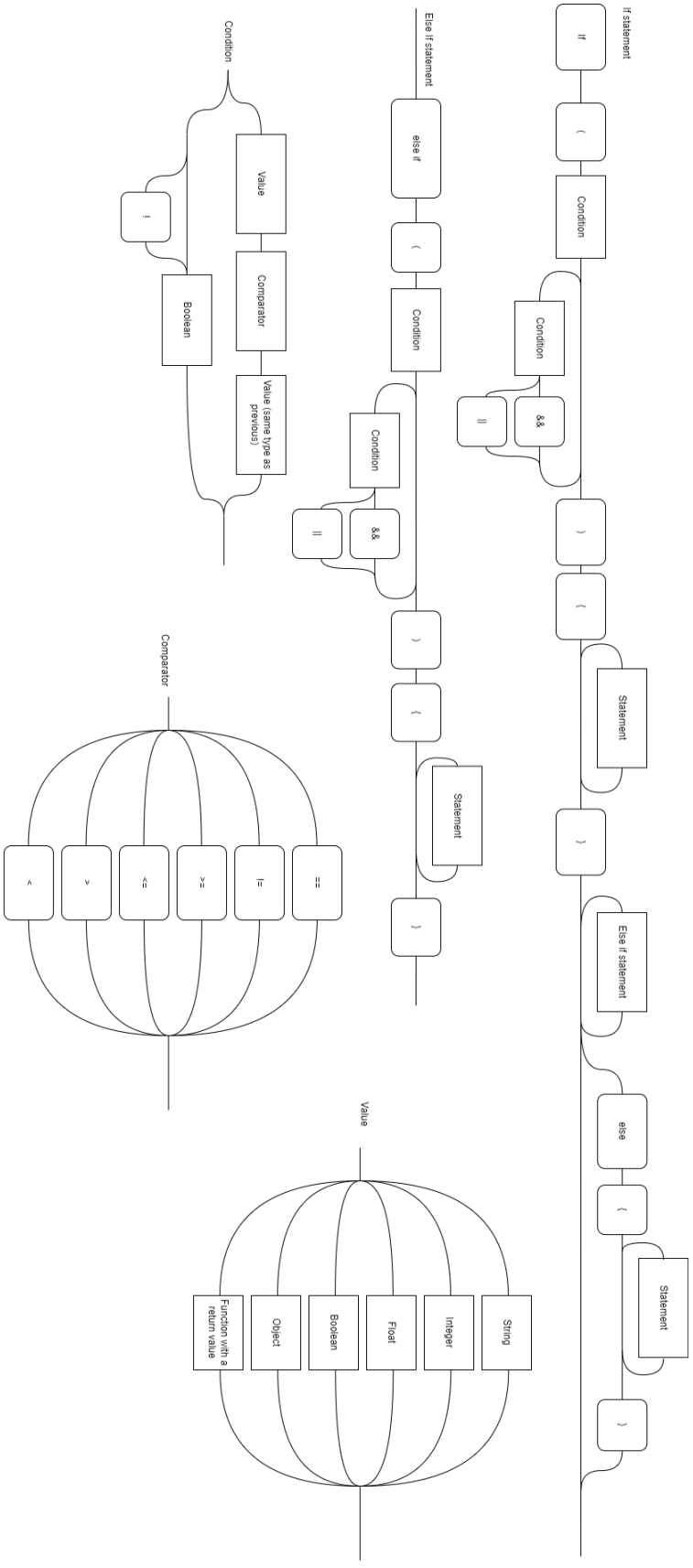
The switch statement:

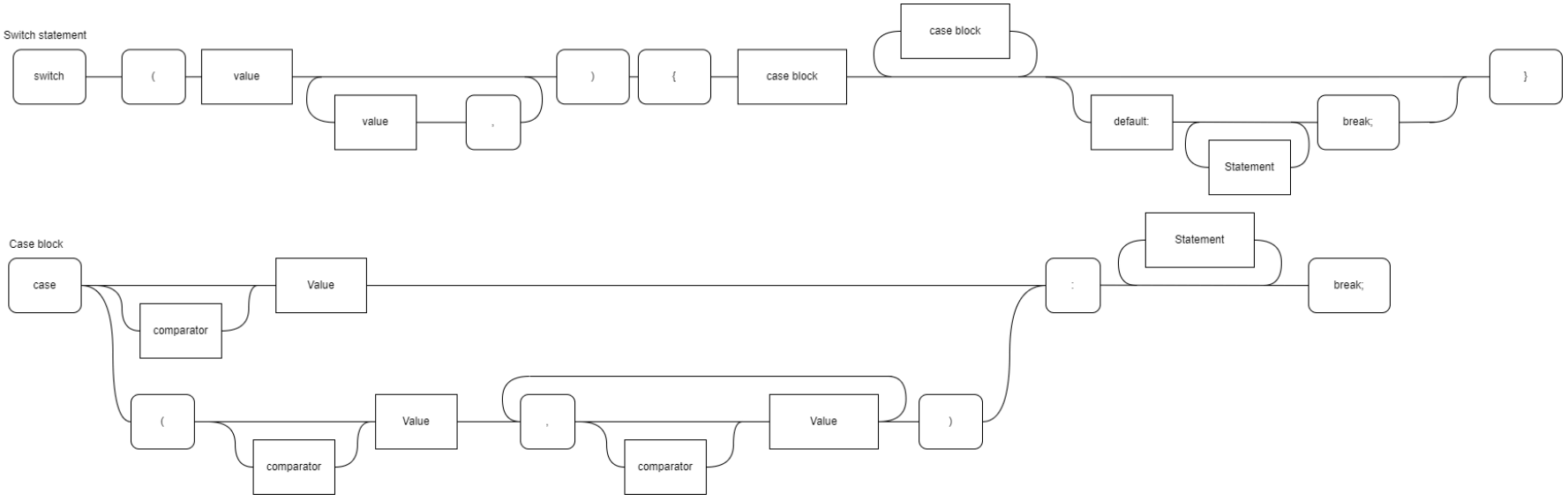
SwitchStatement = switch“(”<value>{,<value>}“){”<CaseBlock>{<CaseBlock>} [default:{<statement>}break;]“}”

CaseBlock = case([comparator]<value>|“(”[comparator]<value>,[comparator]<value> {,[comparator]<value>}“)”):<statement>break;

Railroad diagram vers

For better resolution, please download the image from the source files on GitHub:
(https://github.com/TYProjectUploader/Nyanto/tree/main/Major%20project%20theory%20images)




Switch statement
switch
(
value
value
,
)
{
case block
case block
default:
Statement
break;
}
Case block
case
comparator
Value
(
comparator
Value
,
comparator
Value
)
:
Statement
break;

Test data

Code tested (irrelevant minor things like playing a sound effect and changing colour of cats have been omitted for clarity sake since they are not being tested):

Note: a “GameObject” is a Unity record data type that holds the components and data associated with each object in the game.

```
    private IEnumerator CombineCats(GameObject cat1, GameObject cat2)
    {
        //Destroy collided objects only occurs after rest of frame logic is
done
        Destroy(cat1);
        Destroy(cat2);
        //if watermelon cat do nothing
        if (cat1Index == CatGenerator.instance.Cats.Length -1)
        {
            Debug.Log("Watermelon combine!!!");
        }
        else
        {
            //if not final cat, spawn a cat of next index
            Vector3 collisionPosition = (cat1.transform.position +
cat2.transform.position)/2;
            GameObject combinedCat =
Instantiate(CatGenerator.instance.Cats[cat1Index+1]) ;
            combinedCat.GetComponent<Rigidbody2D>().simulated = true;

            //clamp the spawn position to prevent larger fruits from getting
stuck in the side of walls when spawning
            //get width of cat to be spawned and clamp spawn position within
bounds
            CompositeCollider2D combinedCatCollider =
combinedCat.GetComponent<CompositeCollider2D>();
            float catWidth = (combinedCatCollider.bounds.max.x -
combinedCatCollider.bounds.min.x)/2;
            //top bound not considered coz no effect
            bottomBound = spawnBounds.min.y + catWidth;
            leftBound = spawnBounds.min.x + catWidth;
            rightBound = spawnBounds.max.x - catWidth;

            collisionPosition.y = Mathf.Clamp(collisionPosition.y,
bottomBound, topBound);
            collisionPosition.x = Mathf.Clamp(collisionPosition.x,
leftBound, rightBound);
```

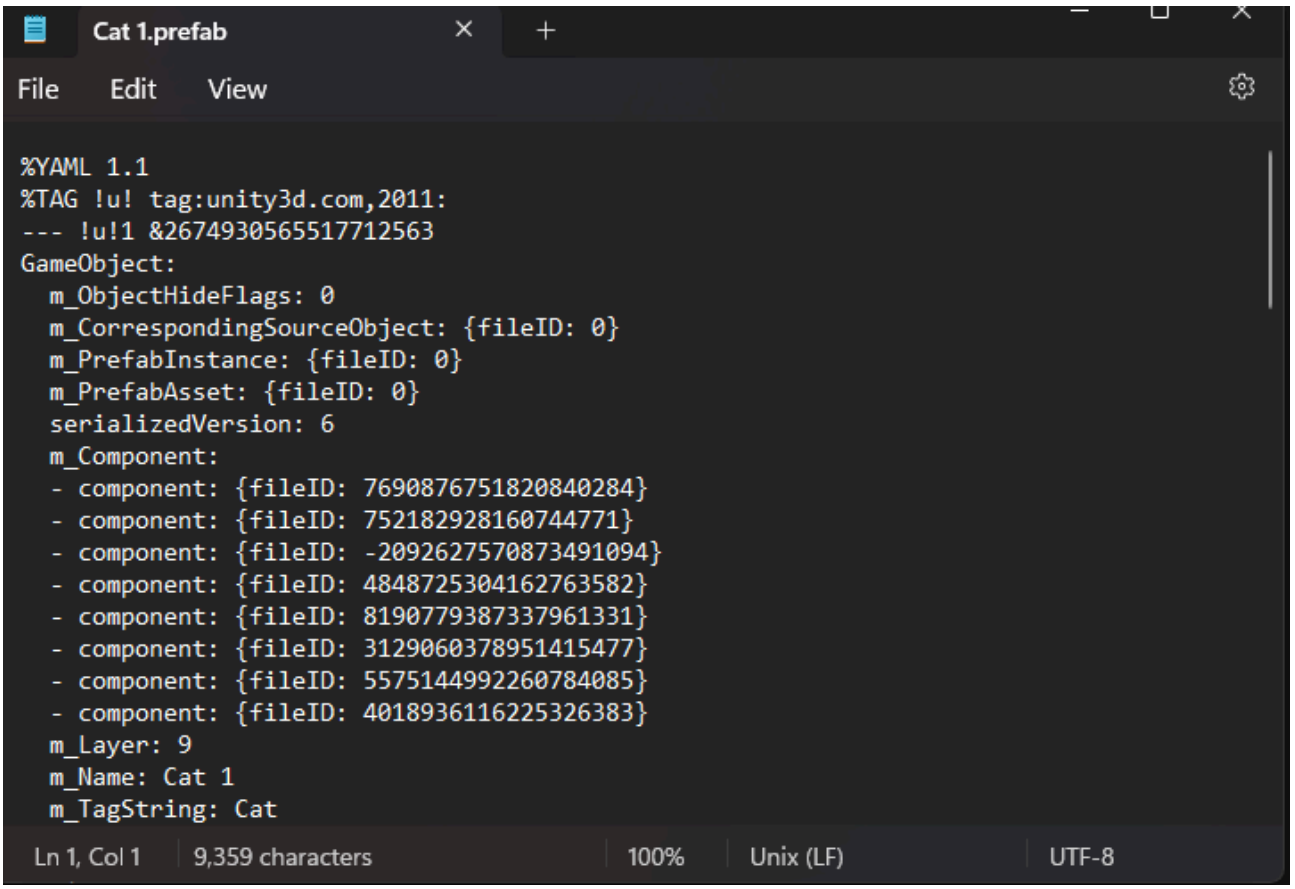
```
            //spawn the cat in at the position calculated
            combinedCat.GetComponent<Rigidbody2D>().position =
collisionPosition;
            combinedCat.transform.position = collisionPosition;

            //To reduce overlap when spawning all overlapping colliders have
their rigidbody receive a force away from each other
            Collider2D[] colliders =
Physics2D.OverlapCircleAll(collisionPosition, catWidth);
            foreach (Collider2D collider in colliders)
            {
                Rigidbody2D rb = collider.GetComponent<Rigidbody2D>();
                if (rb != null)
                {
                    Vector2 repulsionDirection = (rb.position -
(Vector2)combinedCat.transform.position).normalized;
                    rb.AddForce(repulsionDirection * REPULSION_FORCE,
ForceMode2D.Impulse);
                }
            }
        }
    }
```

Test was done by creating a script that takes any 2 GameObjects placed in the editor to the Serialize field or dragging prefabs into the text editor to instantiate them manually. (Nyanto project\Assets/Main Scripts/Normal & Shared scripts/Game scripts/Catcombiner test.cs)

```
1 reference
[SerializeField] public GameObject Catd1;
1 reference
[SerializeField] public GameObject Catd2;
// to test CombineCats this dummy module was created so that cat prefabs can be directly injected as parameters
//create an empty game object and attach this script to use to test
0 references
void Start()
{
    GameObject cat1 = Instantiate(original: Catd1);
    GameObject cat2 = Instantiate(original: Catd2);
    CombinationManager.instance.CombineCats(cat1: cat1, cat2: cat2);
}
```

Test data for the CombineCats module(Pseudocode can be found here if easier to follow than the above).
Note that the “data value” refers to the cloned record for the given cat. This information is stored as serialised binary but the base .prefab record file is written in YAML before compilation, making it human-readable.
It is not practical for me to put the entire record of each cat1 or cat2 here so I refer to them simply as “cats[catnumber]” e.g. cats[1] means I’m referring to a cloned “Cat 1.prefab” record file (each clone has a different fileIDs that differentiates them). The base records can be found under: Nyanto project\Assets\Sprites and prefabs\Cat Prefabs\Cat Prefabs. I store a copy of these base records in an array in the cat generator to create clones. These clones are created by sending the base record as a parameter to the Unity “Instantiate” function.
The base record file for each cat looks like this just fyi:



The test data

<table>
<tr><th>Input & description of test + boundaries being tested</th><th>Expected output description</th><th>Actual output in the game</th><th>Output evaluation/ error response</th></tr>
<tr>
<td>Test:
2 cats of 1st tier combined
<table>
<tr><td>Inputs</td><td>cat1</td><td>cat2</td><td>colliders[]</td></tr>
<tr><td>Data value</td><td>cats[1]</td><td>cats[1]</td><td>null</td></tr>
</table>
Boundary being tested:
Combination of lowest value cats

What it looks like in game:

</td>
<td>A cat of the 2nd tier should be generated in the averaged position of the previous 2 cats. This is done by cloning the base record of a cat of next tier.</td>
<td>
Correctly spawned cat of 2nd tier in the averaged position of previous 2 cats. This is done by Instantiate returning a cats[2]</td>
<td>correct</td>
</tr>
<tr>
<td>Test:
2 cats of last tier combined
<table>
<tr><td>Inputs</td><td>cat 1</td><td>cat 2</td><td>colliders[]</td></tr>
<tr><td>Data value</td><td>cats[11]</td><td>cats[11]</td><td>null</td></tr>
</table>
Boundary being tested:
Combination of highest value cats

What it looks like in game:

</td>
<td>Both cats are deleted from the container (records deleted) and nothing happens after.</td>
<td>
Both of original cats were deleted correctly.</td>
<td>correct</td>
</tr>
</table>

Test: A pair of cats colliding against the side of a wall on left or right (tested with cats of index 3 and left wall) Boundary being tested: Position of the combined cat being clamped correctly otherwise, the combined cat would spawn partially in the wall. What it looks like in game (orange outline is the box walls): 	The original spawn position would be the averaged position of the 2 tier 4 cats whic would cause the spawned 5th tier cat to be partially within the box like this:  However, the spawn position should be clamped so that the entire cat can spawn within the box's bounds.	 The newly spawned cat is correctly clamped inwards from the left wall so that it spawns within the bounds of the box. Instantiate returns a cats[5] and its position is correct. Both of original cats were deleted correctly.	**correct**

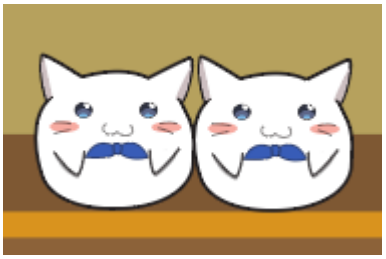
Inputs	cat 1	cat 2	colliders[]
Data value	cats[4]	cats[4]	null

Test:
A pair of cats colliding on the bottom wall by rolling into each other. (tested with cats of tier 4)

Inputs	Input 1	Input 2	colliders[]
Data value	cats[4]	cats[4]	null

Boundary being tested:
Position of the combined cat being clamped correctly otherwise, the combined cat would spawn partially in the wall.

What it looks like in game (orange outline is the box walls):



The original spawn position would be the averaged position of the 2 4th tier cats, but the spawned combined cat of index 4 would be partially within the box, like this:



But the spawn position should be clamped so that the entire cat is spawned just within the bounds of the box.



The newly spawned cat is correctly clamped upwards from the bottom wall to spawn within the box's bounds.

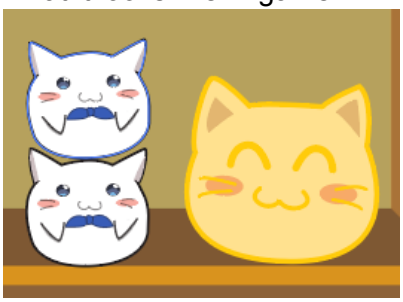
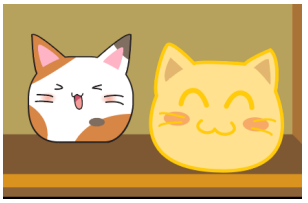
Instantiate returns a cats[5] and its position is correctly adjusted to not be in the box.

Both of original cats were deleted correctly.

correct

<table>
<tr>
<td>Test:
A pair of cats being combined with a cat nearby but outside the radius of the newly spawned cat.(tested with cats of tier 4 combining and index 6 cat nearby)

<table><tr><td>Inputs</td><td>cat 1</td><td>cat2</td><td>colliders[]</td></tr><tr><td>Data value</td><td>cats[4]</td><td>cats[4]</td><td>null</td></tr></table>
Boundary being tested:
Other cats in the radius of the newly spawned cat are correctly being pushed away.

What it looks like in game:
</td>
<td>The 2 cats of tier 4 should just combine as usual and the tier 6 cat is unaffected.</td>
<td>
Output is as expected, with the new 5th tier cat spawning in the average position of the two tier 4 cats. Tier 6 cat is not moved at all

Instantiate returns a cats[5] and its position is correct.

Both of original cats were deleted correctly.</td>
<td>correct</td>
</tr>
</table>

Test:
A pair of cats being combined with a cat within the radius of the newly spawned cat is meant to spawn. (tested with cats of tier 4combining and tier 6 cat in spawn radius)

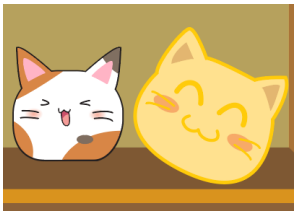
Inputs	cat1	cat2	colliders[]
Data value	cats[4]	cats[4]	cats[6].collider

Boundary being tested:
Other cats in the radius of the newly spawned cat correctly being pushed away.

What it looks like in game:



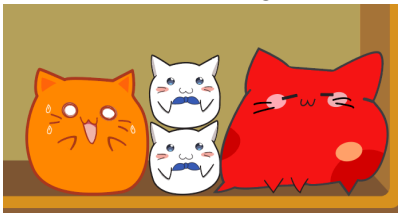
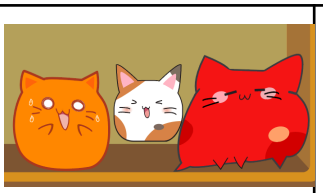
The 2 cats of tier 4 should combine then push the 6th tier cat away to the right as it is within the radius of the newly spawned tier 5 cat.



The tier 6 cat is correctly pushed away from where the tier 4 cat spawns to make room for it.

Instantiate returns a cats[5] and its position is correct.
Both of original cats were deleted correctly.

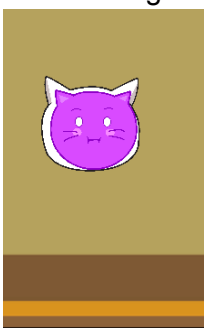
correct

Test: A pair of cats being combined with cats below and to the sides of where the new cat is meant to spawn. (tested with cats of tier 4 combining with tier 7 and 8 in spawn radius) **Inputs** / **cat1** / **cat2** / **colliders[]** Data value / cats[4] / cats[4] / cats[7].collider, cats[8].collider Boundary being tested: Other cats in the radius of the newly spawned cat are correctly pushed away when other cats may not have space to move. What it looks like in game: 	The 2 cats of tier 4 should combine and push the tier 7 cat to the left and tier 8 to the right. However, because the tier 8 cat is unable to move due to being against the wall, it will be slightly jostled. The Unity physics engine will resolve the tier 5 cat being inside tier 8 cat in the next frame or so by changing its position to be just outside of the collider of the tier 8 cat as it is larger.	 Output is as expected. Index 6 cat is pushed to the left while index tier 8 has been slightly jostled, and tier 5 cat has correctly spawned without issues. Instantiate returns a cats[5] and its position is correct. Both of original cats were deleted correctly.	**correct**

Test:
A pair of 2 different cat records being sent to the module using the script I set up.

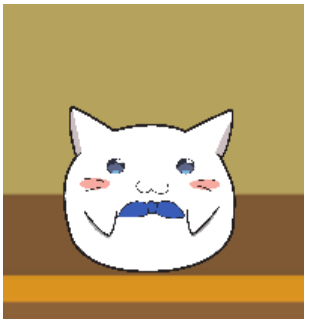
Inputs	cat1	cat2`	colliders[]
Data value	cats[3]	cats[4]	null

What it looks like in the frame forcibly instantiating them before being combined:



Boundary being tested:
The scenario of two different cat GameObject records being passed which can’t happen in normal operation

Based on the logic of the code a cats[4] should be be returned by the Instantiate call since it's based on the index associated with cat1 +1. The CombineCats module in practice will never be able to do this since the records input would be of the same type but by forcing these inputs it will.



Output is as expected with cats[4] being returned from the Instantiate.

Both of original cats were deleted correctly.

Technically correct

<table>
<tr>
<td>Test:
A cat game object and a non-cat GameObject being sent to the module using the script I set up with a cat1 as the non-cat GameObject since it doens’t have an index field.

<table><tr><td>Inputs</td><td>cat1</td><td>cat2</td><td>colliders[]</td></tr><tr><td>Data Values</td><td>Bomb GameObject from explosive mode</td><td>cats[3]</td><td>null</td></tr></table>
What it looks like in game after forcibly instantiating the GameObjects to send into the module:

Boundary being tested:
The scenario of two different cat GameObject records being passed into the module which can’t happen in normal operation</td>
<td>Some kind of error from Unity that will halt execution as it won’t be able to find an index from the bomb GameObject.</td>
<td>
No error was thrown and a cat of tier 2 was spawned with the bomb and cats[3] destroyed as cat1’s index just defaulted to 0 when it shouldn’t have one… This is likely something on Unity’s end I can’t control.</td>
<td>UNEXPECTED ERROR</td>
</tr>
</table>

Error checking

Stubs

Stubs have since been changed into actual subroutines with processing within the source code. The following are examples of what was previously there to test higher-level modules:

Example 1:

Stub for updating the movement boundary of the player when a new cat is spawned by calling updateMovementBoundary in PlayerMovement.cs from CatGenerator.cs which are both found in the path: Assets/Scripts/Normal & Shared scripts/Game scripts/

Here the logic for updating the boundary based on the width of the current cat in the player's hand has not been developed, so the boundary is just set to specific values to verify it is being correctly called, and a statement is printed in the console to indicate the stub was successful.

```
public void SpawnCat()
    {
        //spawn cat at the location of an empty game object called
catspawner
        currentCat = Instantiate(nextCat, spawnerPosition.transform);
        //get size of cat radius to update movement boundary
        CompositeCollider2D currentCatCollider =
currentCat.GetComponent<CompositeCollider2D>();
        float catWidth = (currentCatCollider.bounds.max.x -
currentCatCollider.bounds.min.x)/2;
        PlayerMovement.instance.UpdateMovementBoundary(catWidth);
        currentCat.SetActive(false); //hide the newly spawned cat
        //once next cat instantiated generate the next cat
        GenerateRandomcat();
    }
```

Stub:

```
public void UpdateMovementBoundary(float catWidth) //update movement
boundary based on size of cat
    {
        leftBound = -2.8
        rightBound = 2.8
        Debug.Log("Movement Boundary updated");
    }
```

Example 2:
When game over is triggered, a screenshot of the current game state is meant to be taken and displayed in an image box on the end screen menu (replacing the default white image). While the logic for capturing the screenshot (CaptureScreenshot()) is not developed, a predetermined image file was loaded to replace the default white image instead to simulate GameOver being called when game over is triggered correctly during testing.
Path: Assets/Scripts/Normal & Shared scripts/Game scripts/GameManager.cs

```
public void GameOver()
    {
        AudioManager.instance.PlaySFX(AudioManager.instance.SFXGameOver);
        gameOver = true;
        DisableCatPhysics();
        CaptureScreenshot();
        FadeOutGame();
    }
```

Stub:

```
Private void CaptureScreenshot()
    {
      gameScreenShot.sprite
=Resources.Load<Sprite>("Resources/Screenshot/gameOverScreenShot.png");
    }
}
```

(This was the placeholder gameOverScreenShot.png used)



Flags

Multiple public boolean variables have been used as flags to confirm that certain subroutines have already occurred. These boolean dictates whether other subroutines are allowed to run and can be checked in the Unity Editor while in play mode to view when flags are triggered in real time. The ability to view it in the editor acts similarly to a watch expression, thus allowing errors of conditions not being met/detected to be traced.

Example ColliderInformer.cs (3 flags in one script):

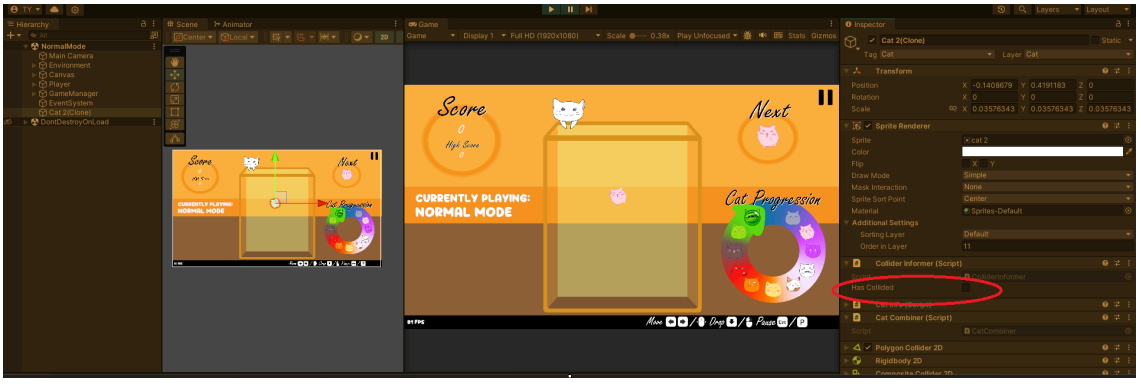
The following script is attached to every cat prefab. When a given cat prefab not generated from other cats combining collides with an object for the first time, it sets the flag for being able to drop as true to indicate conditions for being able to drop the next cat has been met and its own flag “hasCollided” that it has already collided with something to prevent itself from running again and spawning another cat. However, both flags “hasCollided” and “triggeredFromCatCombine” must be false for the above to occur. “triggeredFromCatCombine” is a flag that is set to true by CombinationManager.cs when a cat prefab is generated from a cat combination rather than by the player to prevent the new cat prefab calling SpawnCat.

Path: Assets\Scripts\Normal & Shared scripts\Cat related scripts\ColliderInformer.cs

```
void OnCollisionEnter2D(Collision2D collision)
    {
        if (!hasCollided && !triggeredFromCatCombine)
        //condition check necessary or else causes bug where 2 cats are spawned if current cat collides with 2 objects in the same frame or from combine collision
        {
            hasCollided = true;
            PlayerMovement.instance.canDrop = true; //enable spawning again
            CatGenerator.instance.SpawnCat(); //spawn next cat when fruit has collided
        }
        StartCoroutine(DestroyAfterDelay(3f)); //destroy the script after 5s to improve performanc and account for scripts being created when cats combine (when triggeredFromCatCombine is true)
    }
```

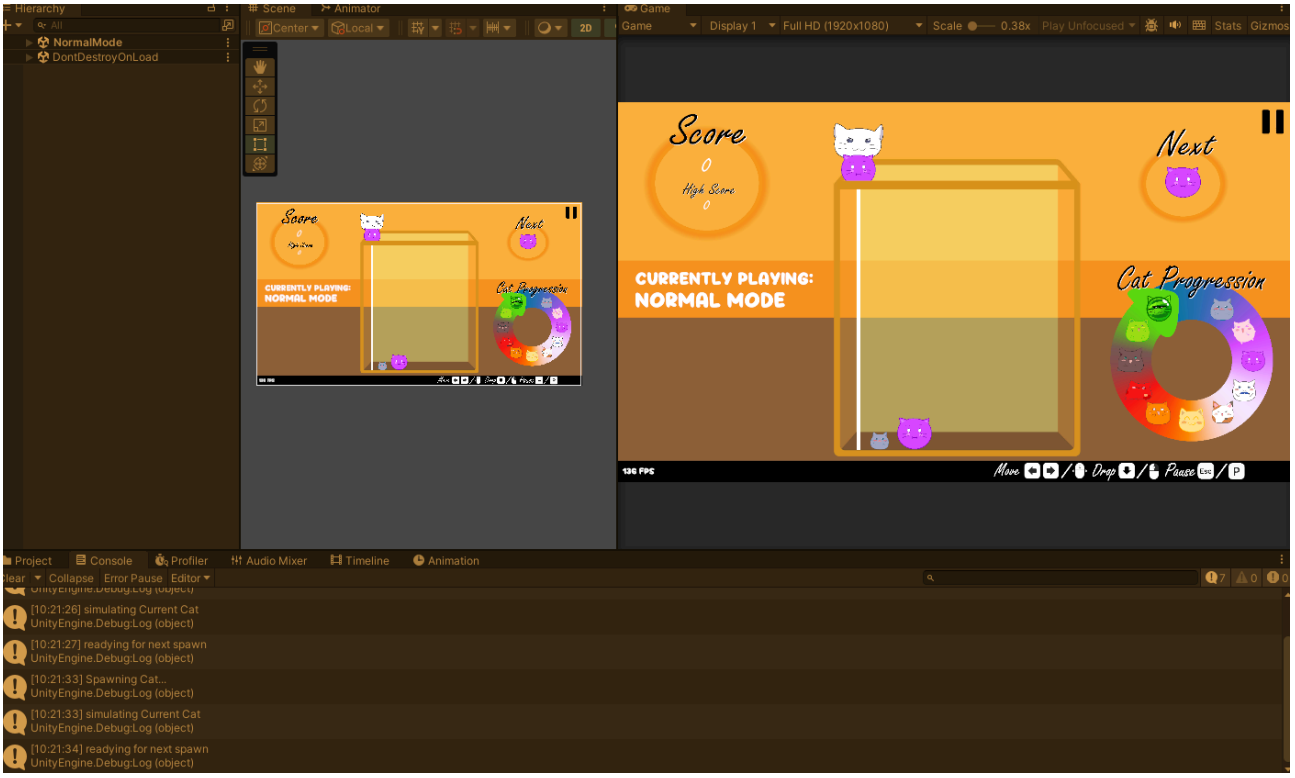
How it looks in the Unity Editor:

A cat prefab is dropped and the flag of hasCollided is not set to true yet, but once it hits the ground the box becomes ticked indicating the flag has been set to true.



Debugging output statements

These statements help isolate the source of errors by allowing for determination of which subroutines have been called and monitor the flow of execution or display the contents of variables at certain stages during execution. They are often used over watch expressions due to their ease of implementation and ability to provide immediate and traceable feedback during execution without having to pause execution or step through it line by line. Unity's debugging output statements can be implemented as code using: Debug.log("error message here");



(Output statements visible in bottom left)

Example 1:
A debug statement that was printed to the Unity editor console indicating each frame a cat stays inside the boundary that determines when game over in triggered. When it has stayed over a certain time, the game ends and a statement printing gameover is triggered. However, if the cat exits before the time is up, then the debug statement saying it has exited is printed.
Path: Assets\Scripts\Normal & Shared scripts\Game scripts\GameOverDetector.cs

```
    void OnTriggerStay2D(Collider2D collider)
    {
        //when a cat stays in collider over the timetillgameover the game
ends
        if (collider.gameObject.GetComponent<Rigidbody2D>().simulated &&
!GameManager.instance.gameOver && collider.gameObject.CompareTag("Cat"))
        {
            timer += Time.deltaTime;
            if (timer > timeTillGameOver)
            {
                GameManager.instance.GameOver();
                StartCoroutine(FlashGameOverCat(collider.gameObject));
                Debug.Log("gameover");
            }
            else
            {
                Debug.Log("Cat in game over bounds");
            }
        }
    }
    void OnTriggerExit2D(Collider2D collider)
    {
        timer = 0f;
        Debug.Log("exited");
    }
```

Example 2:
To ensure the singleton was working correctly such that there weren't multiple calls to HandleCollision in the same frame which was causing the bug of infinite combinations leading to Unity crashing as outlined in the 1st runtime error here, debug statements allowed the execution of the following processes to be tracked thus verifying that when there were multiple calls to the function they weren't simultaneously processed by the HandleCollision logic.
Path: Assets\Scripts\Normal & Shared scripts\Game scripts\CombinationManager.cs

```
public void HandleCollision(GameObject catPrefab)
{
    …
    Debug.Log("handling collision");
    if (cat1 == null)
    {
        cat1 = catPrefab;
        cat1CatInfo = cat1.GetComponent<CatInfo>();
        cat1Index = cat1CatInfo.catIndex;
        cat1Value = cat1CatInfo.combineValue;
        Debug.Log("cat1 stored");
        return;
    }
    else if (cat2 == null)
    {
        cat2 = catPrefab;
        cat2CatInfo = cat2.GetComponent<CatInfo>();
        cat2Index = cat2CatInfo.catIndex;
        Debug.Log("cat2 stored");
    }
    …
    Debug.Log("clearing stored cats");
    cat1 = null;
    cat2 = null;
}
```

Desk check of one module

Did desk check for the CombineCats Module (same one used for flowchart/pseudocode). Example desk check is for the following starting variables for 2 cats of the 2nd index (cat1 & cat 2) combining close to the wall that would make the combined cat spawn inside the wall while there are 2 other different cats (found in array named colliders at the collider positions listed) nearby.

Length of cat prefab array	cat 1Index	cat1 Position	cat2 Position	spawnposition	combinedcat	combinedCatColliderMaxx	combinedCatColliderMinx	cat Radius	spawnBounds	Repulsionforce	i	Legnth of colliders	colliderposition	Repulsion direction
11	2	(2,4)	(2,6)						Minx: 1.5 Maxx: 10 Miny: 0 Maxy: 20	1.5		2		
				(2,5)	cats (3)	1	3							
								1	Minx: 2.5 Maxx: 9 Miny: 1 Maxy: 19					
				(2.5, 5)							0		(3,4)	(0.75,-1.5)
											1		(1.7,4.6)	(-1.2,-0.6)

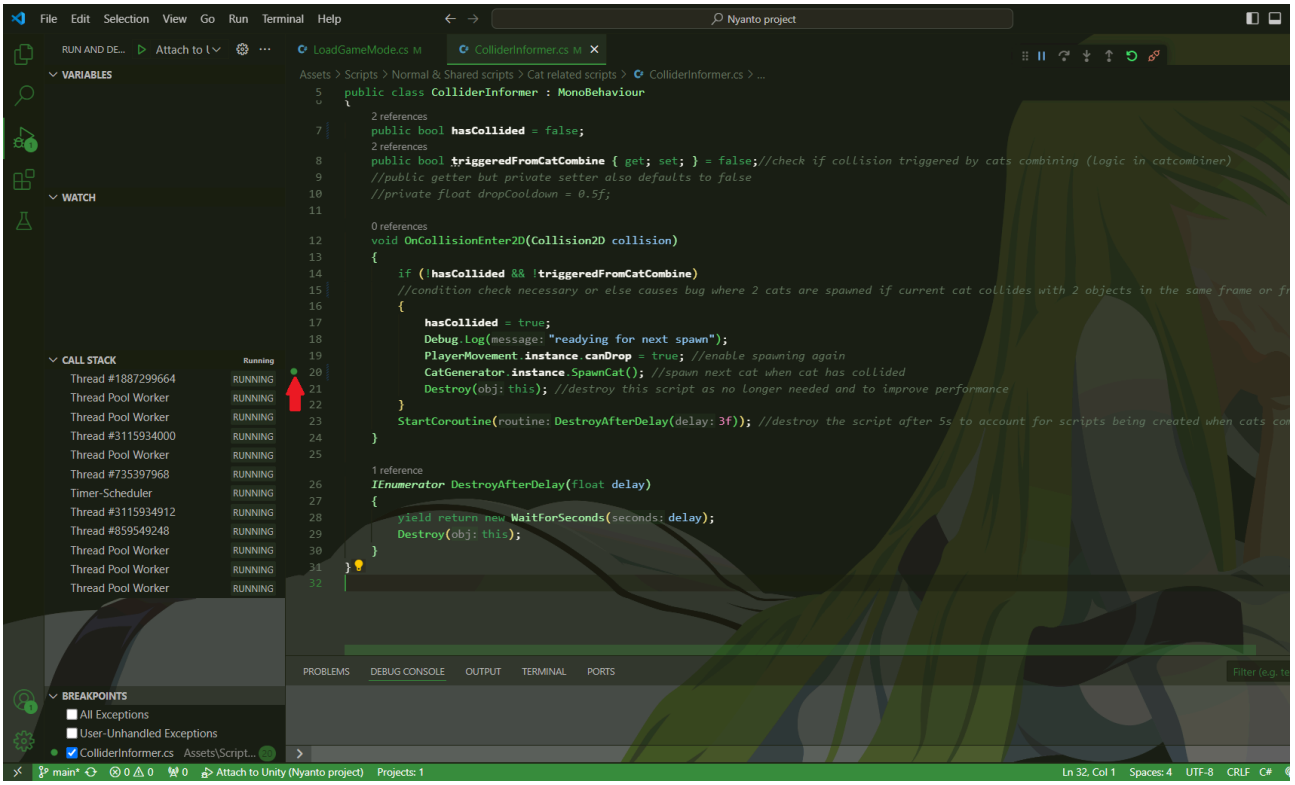
Breakpoint, program traces and single line stepping

Breakpoints

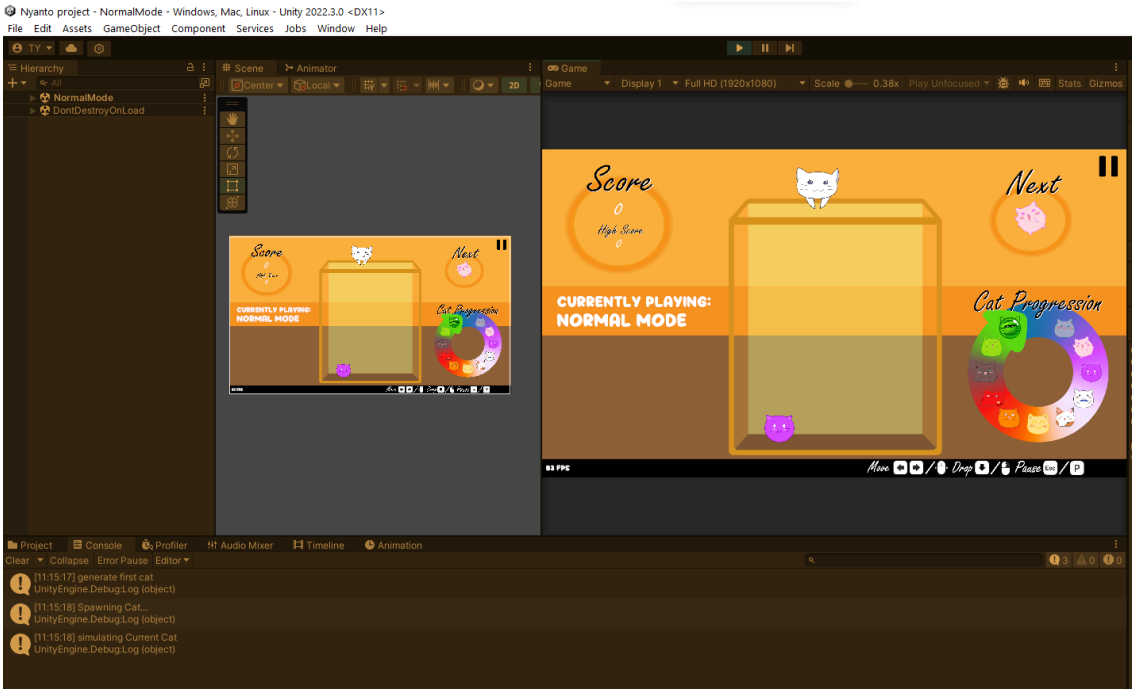
Breakpoints are able to be set using the Unity extension available on VS Code by clicking slightly to the left of a given line number, halting the execution of the game in the Unity Editor.

Example of usage:

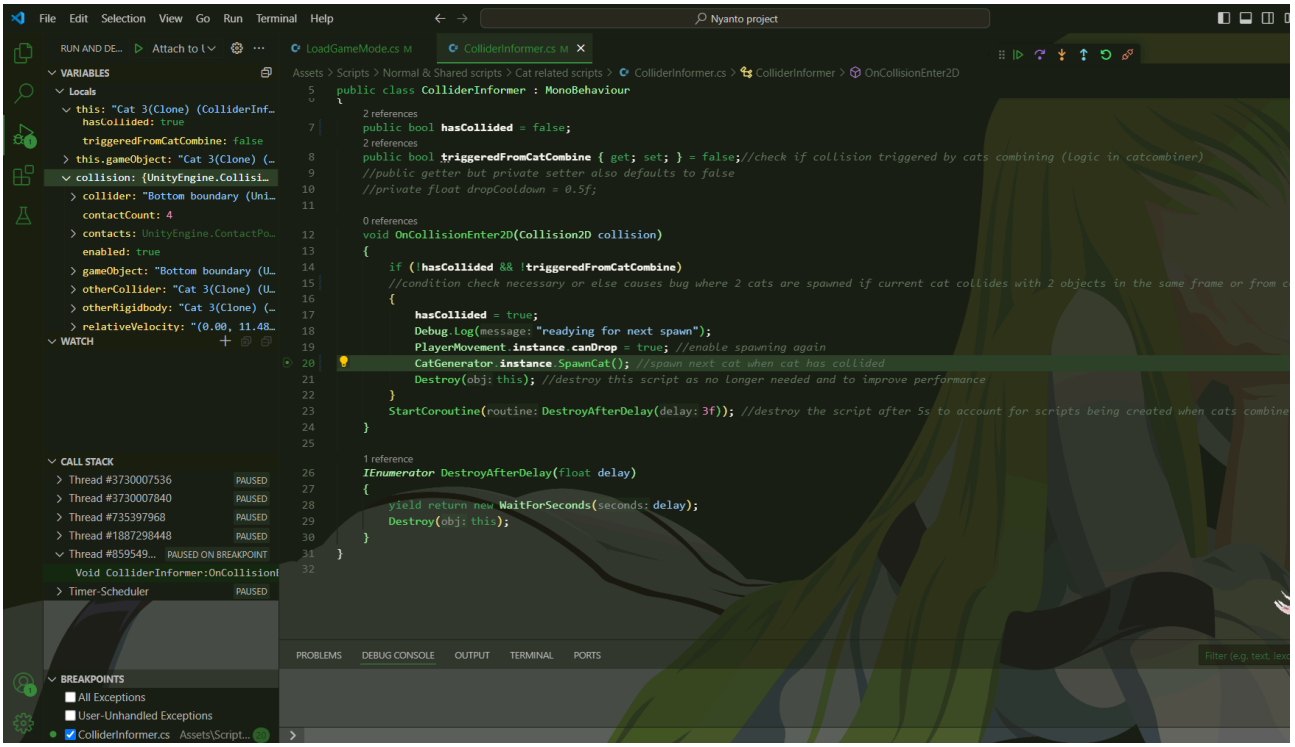
Here a break point is set when a new cat is to be spawned at line 20 as indicated by the arrow.



When this breakpoint is reached, the Unity Editor is frozen in the frame before the frame where the breakpoint is reached as shown below, since the logic after the breakpoint has to be processed before the next frame is drawn.

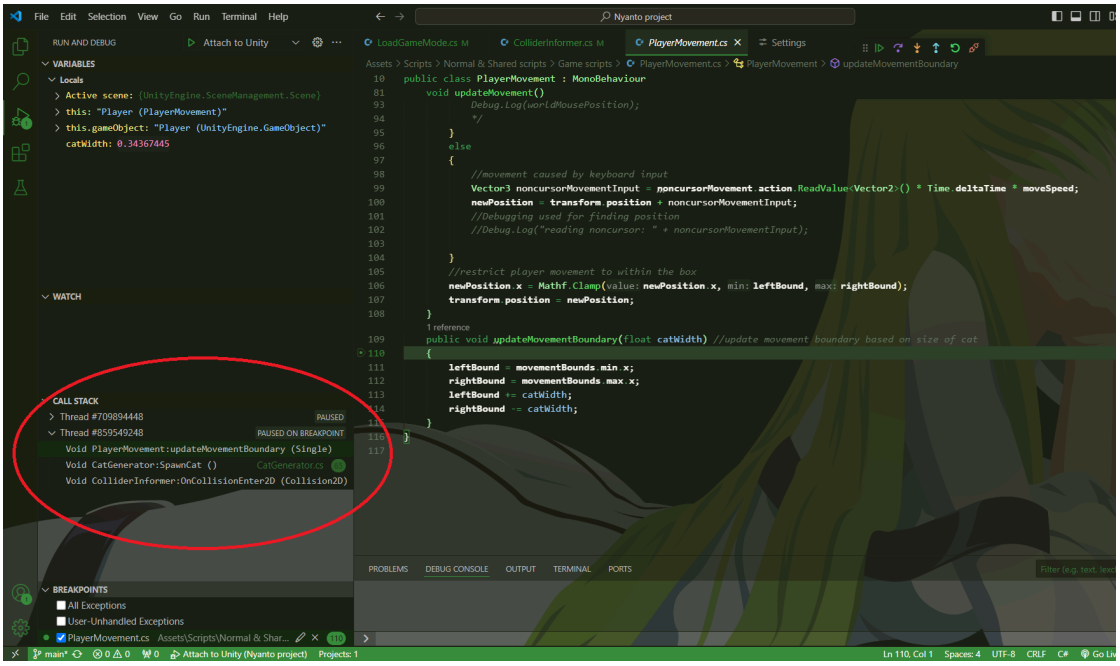


Meanwhile, the code editor highlights where the line where execution has paused allowing for various variables to be examined in the current frame of the game by combining it with program traces which are visible in the left hand side of the editor. Execution can then be continued using the floating window close to the top right corner, allowing for continuation and line stepping.



Program Traces

Other than the ability to view the variables as just pointed out, program traces allow the execution order of functions up to where the break point to be viewed in the call stack.
The example below is showing how the UpdateMovementBoundary call was triggered by the OnCollisionEnter2D function from the ColliderInformer script which has then called the SpawnCat function which then finally calls the UpdateMovementBoundary



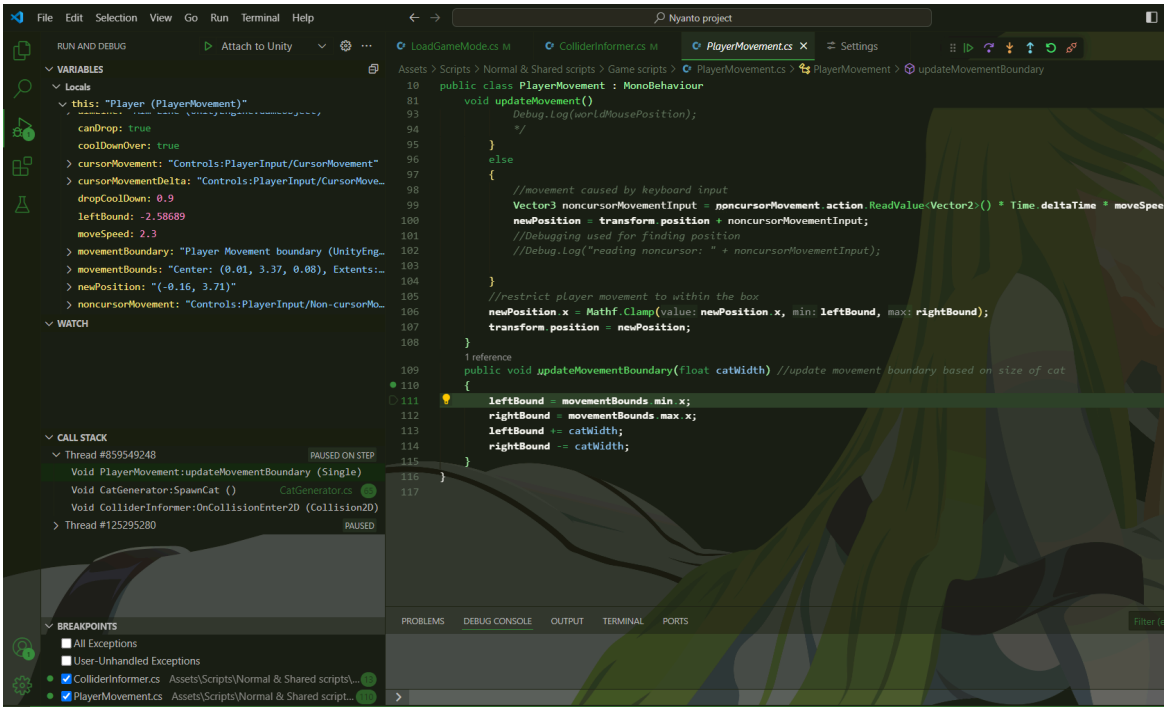
Single Line stepping

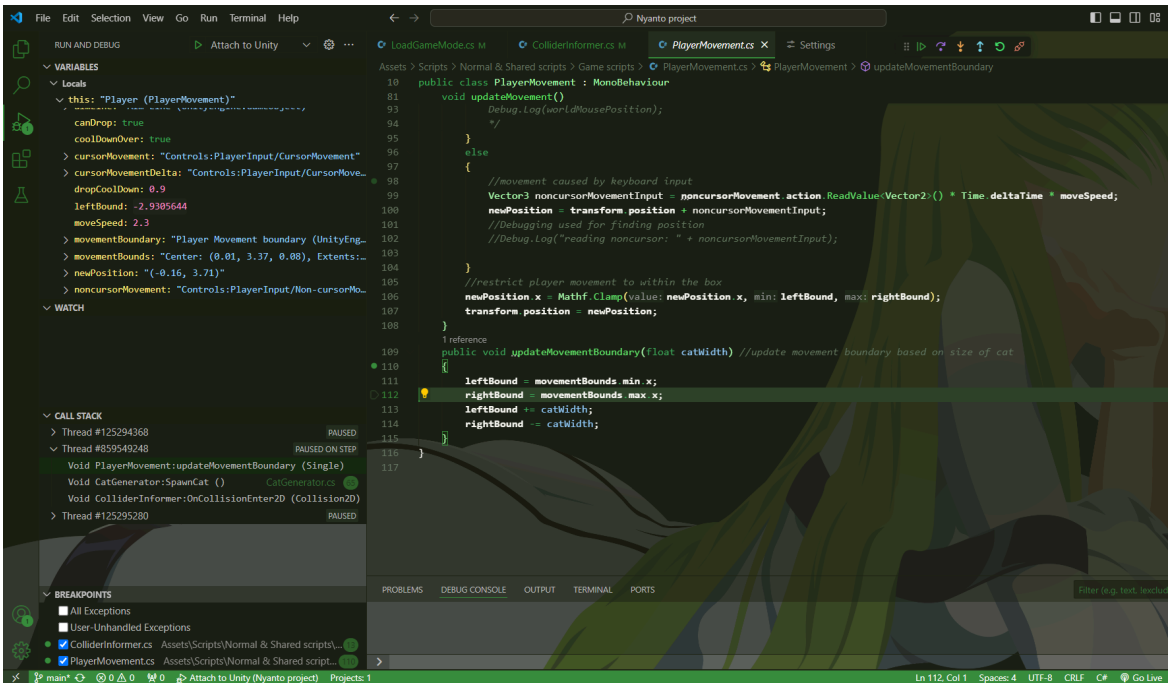
Using these button in VS Code, lines of code can be stepped over and stepped into (purple and orange, respectively).



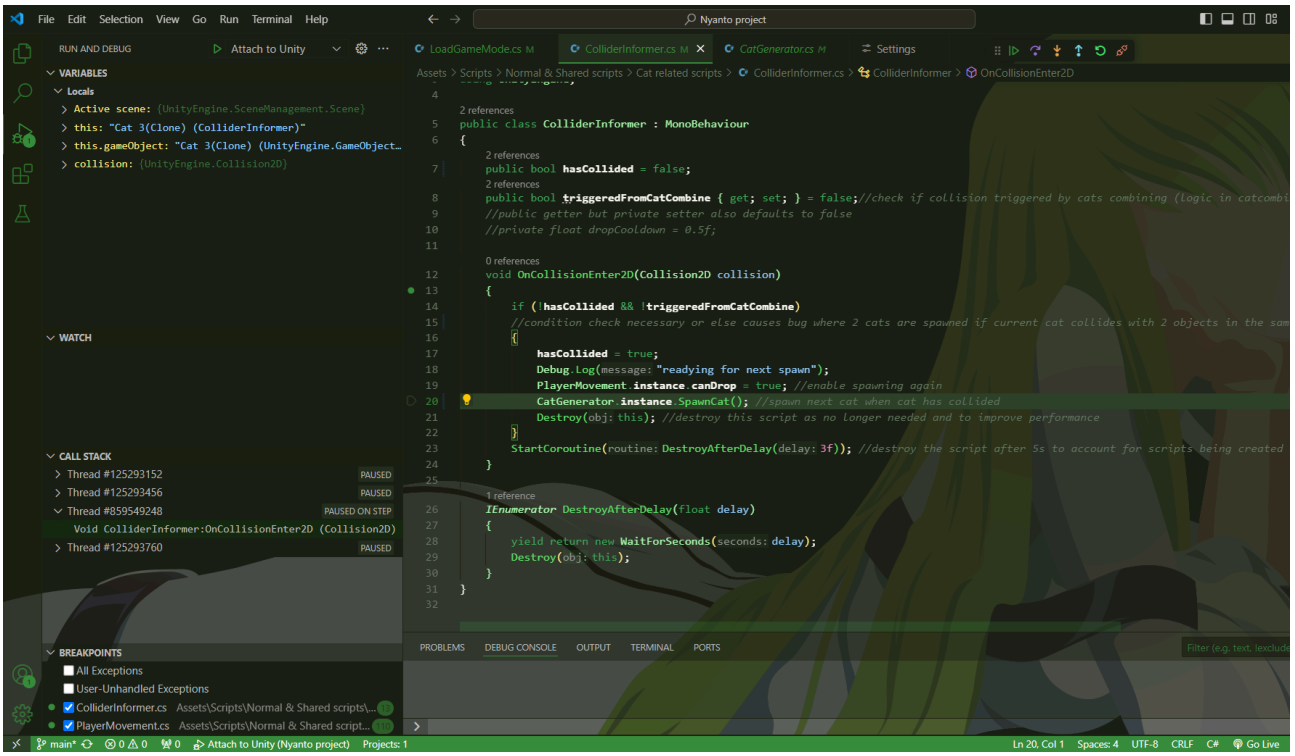
Stepping allows the effect of each line of code to be examined by executing the next line of code before halting execution again.

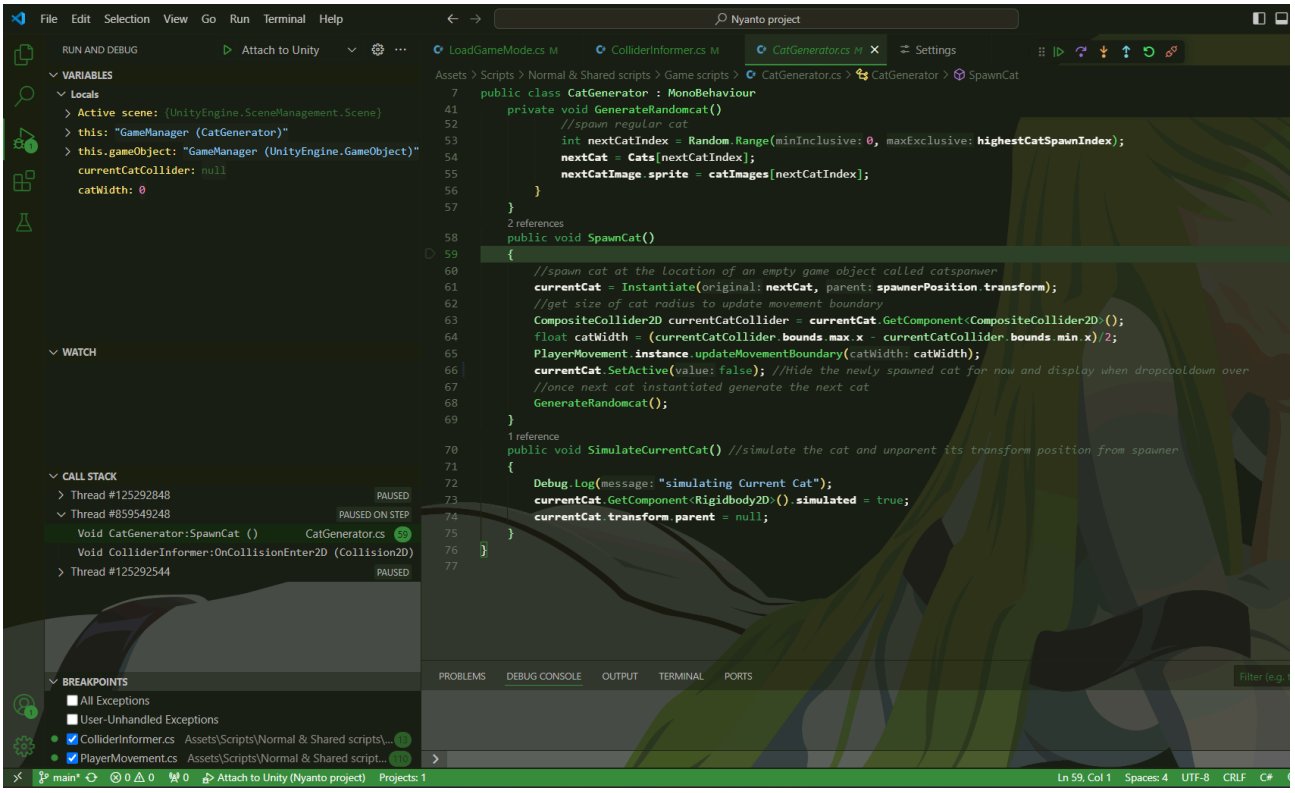
Stepping over is demonstrated in the following two images where the line 111 is stepped over causing the leftBound variable to be updated on the program traces window on the left. This allows me to check that the updateMovementBoundary function is correctly working and easily identify errors if they did exist.





Stepping into is demonstrated in the following two images, where the line 20 is stepped into after the 1st image, causing the editor to instead proceed to the first line of the SpawnCat function rather than line 21 from the 1st image.





Readability of code

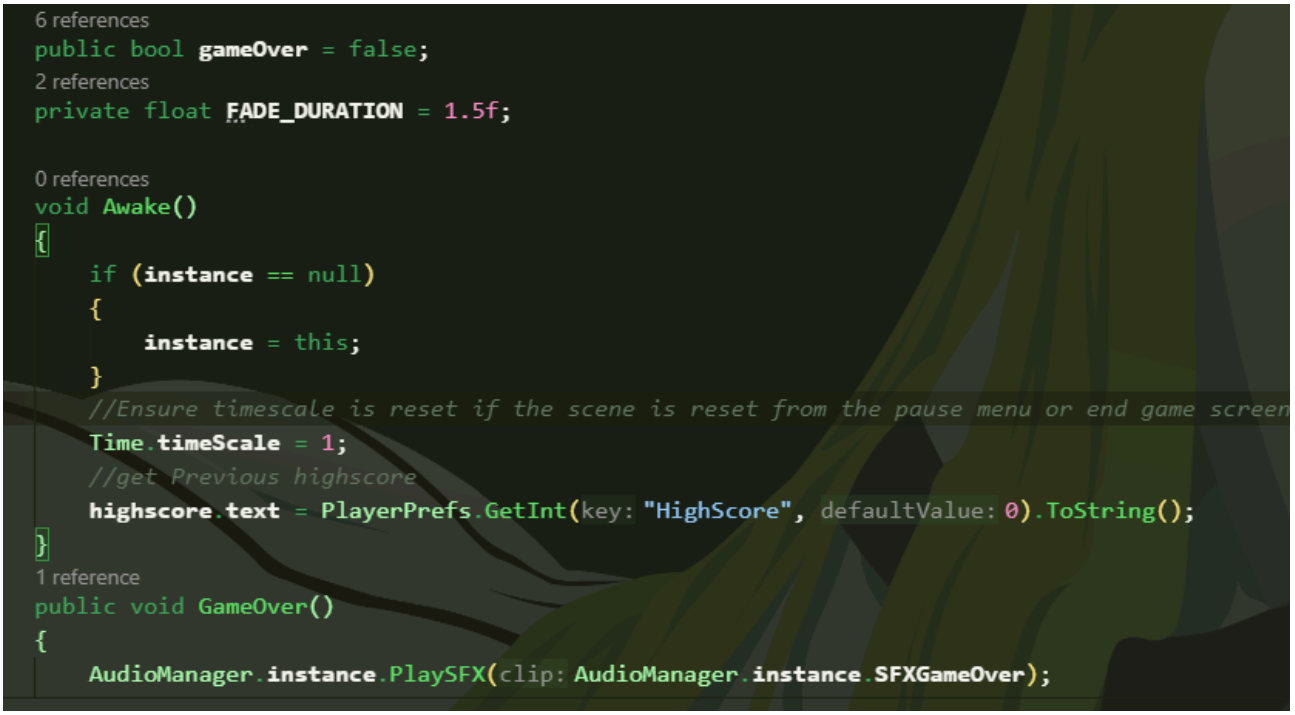
Following naming conventions

In addition to naming variables and functions based on how they are used, they have been appropriately formatted throughout my project using common conventions to allow others to efficiently read code, thus making the code more clean and maintainable.

- PascalCase
  Used for naming methods and classes.
- camelCase
  Used for naming variables
- ALL_CAPS
  Used for naming constants

An example of all elements aforementioned has been taken from:
Assets\Scripts\Normal & Shared scripts\Game scripts\GameManager.cs

```
public class GameManager : MonoBehaviour
{
```



```
    6 references
    public bool gameOver = false;
    2 references
    private float FADE_DURATION = 1.5f;

    0 references
    void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        //Ensure timescale is reset if the scene is reset from the pause menu or end game screen
        Time.timeScale = 1;
        //get Previous highscore
        highscore.text = PlayerPrefs.GetInt(key: "HighScore", defaultValue: 0).ToString();
    }
    1 reference
    public void GameOver()
    {
        AudioManager.instance.PlaySFX(clip: AudioManager.instance.SFXGameOver);
```

Commenting

Commenting is extensively used throughout programs to summarise the purpose of code and/or their logic, especially when they involve subjectively complex procedures and to allow future changes or readings of the code to be efficient.

Example from: Assets\Scripts\Normal & Shared scripts\Game scripts\GameManager.cs

```
    private IEnumerator CaptureScreenshot()
    {
        //create path for screenshot to be saved
        string ssPath;
        //check if done in unity editor or is final build to save the
screenshot in different places
        #if UNITY_EDITOR
        ssPath = Path.Combine(Application.dataPath,
"Resources/Screenshot/gameOverScreenShot.png");
        #else
        ssPath = Path.Combine(Application.persistentDataPath,
"gameOverScreenShot.png");
        #endif
        //capture screenshot of game to display on game over screen
        ScreenCapture.CaptureScreenshot(ssPath);
        //wait for next frame before continuing to ensure screenshot is
written before continuing
        yield return new WaitForSeconds(0.5f);

        //create a texture using the screenshot
        Texture2D texture = new Texture2D(Screen.width, Screen.height);
        byte[] imageData = File.ReadAllBytes(ssPath);
        texture.LoadImage(imageData);

        // Create a Sprite from the Texture2D
        Sprite sprite = Sprite.Create(texture, new Rect(0, 0, texture.width,
texture.height), new Vector2(0.5f, 0.5f));
        //Finally load the sprite
        gameScreenShot.sprite = sprite;
    }
```

Formatting

- Indentation & white space
  Indentation is used to help improve readability of code by visually organising code by logical structure. This can be combined with whitespace to further group logical steps of a module.

The following example demonstrates how similar or related tasks are grouped by separating lines of code with blank lines with indentation clarifying the scope of the if and else statements to improve readability.
Path: Assets\Scripts\Normal & Shared scripts\UI scripts\PauseRelatedButtons.cs

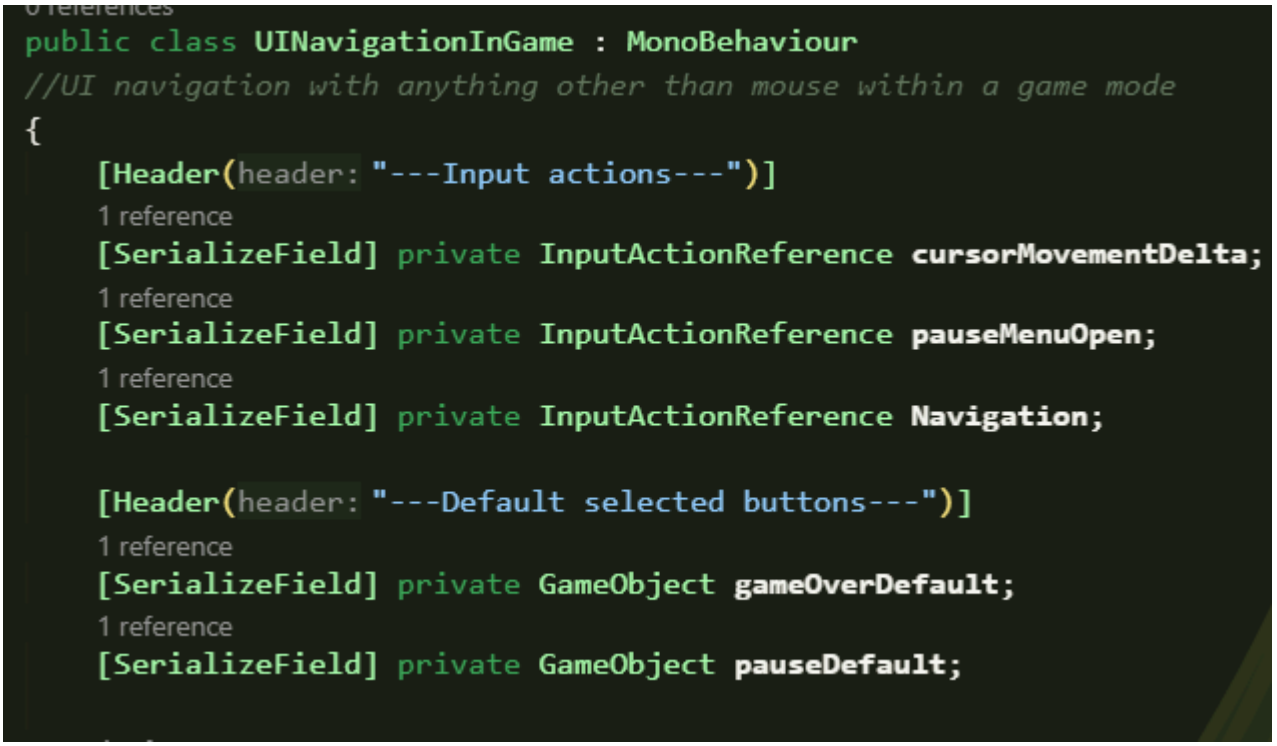


```
public void PauseScreen()
{
    Debug.Log(message: "pause button pressed");
    if (!Paused && !GameManager.instance.gameOver)
    {
        //change out input prompts
        menuPrompts.SetActive(value: true);
        inGamePrompts.SetActive(value: false);

        AudioManager.instance.PlaySFX(clip: AudioManager.instance.SFXButtonClick);
        //if pause screen was set to be inactive hasn't been hidden cancel it to reduce rendering load
        CancelInvoke(methodName: "hidePauseScreen");

        //activate pause screen
        pauseScreen.SetActive(value: true);
        pauseBoardUIAnimator.SetTrigger(name: "Paused");
        Paused = true;
        Time.timeScale = 0;
    }
    else //let player resume by pressing the pause button again.
    {
        Resume();
    }
}
```

- Grouping of variable decelerations
Variables relating to each other can be grouped together in the Unity Editor using headers or within the itself code by using either comments or white space. This has been done for example with the following variables from: Assets\Scripts\Normal & Shared scripts\UI scripts\UI NavigationInGame.cs



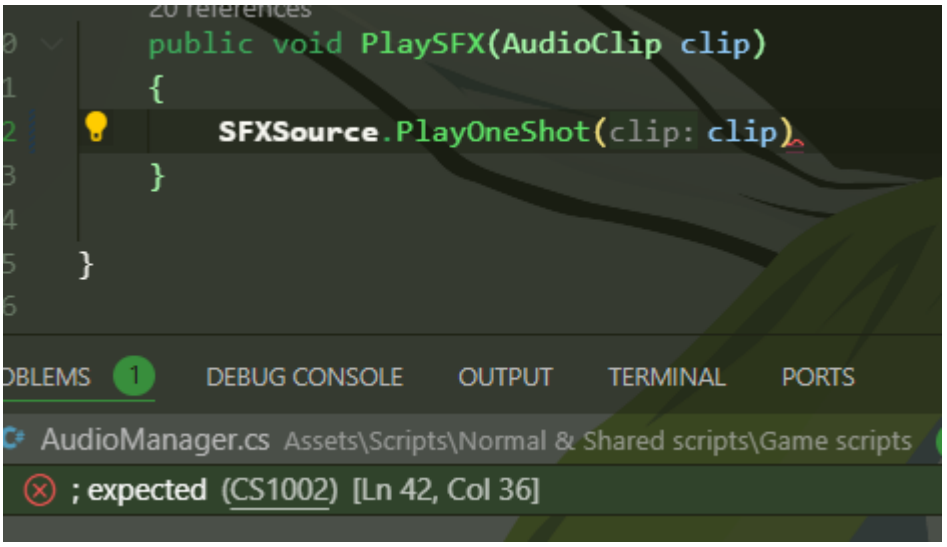
```
public class UINavigationInGame : MonoBehaviour
//UI navigation with anything other than mouse within a game mode
{
    [Header(header: "---Input actions---")]
    1 reference
    [SerializeField] private InputActionReference cursorMovementDelta;
    1 reference
    [SerializeField] private InputActionReference pauseMenuOpen;
    1 reference
    [SerializeField] private InputActionReference Navigation;

    [Header(header: "---Default selected buttons---")]
    1 reference
    [SerializeField] private GameObject gameOverDefault;
    1 reference
    [SerializeField] private GameObject pauseDefault;
```

Discussion of syntax errors, runtime errors, logic errors

Syntax

Due to the immense features provided by modern code editors, syntax errors will pretty much always be caught and diagnosed as code is being written. As code will be prevented from compilation without these fixed, I had little issues with syntax as often it would just be misspellings, missing brackets or semicolons, or incorrect variable assignment for the C# language.
E.g. missing a ";" which indicates the end of a statement in C#.



A notable non-syntax error but similar was a semantic error during development. There would be times when I would be using unity functions from libraries that weren't imported, but this would not be detected by the code editor or compiler, causing some intended features to not work at all. This often happened when I was using UI-related functions.

Logic

Logic errors generated were generally a result of certain gameplay interactions being overlooked or unfamiliarity with a certain Unity game object component type. Errors caused by unfamiliarity with Unity has been mostly left outside this discussion as they were fixed by checking documentation and correct reimplementation.
The following are a few notable logic errors encountered and the way they were resolved (I really mean just a few).

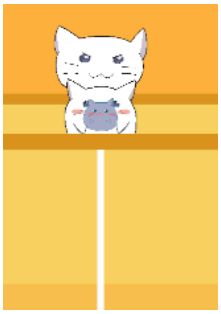
- Cats spawning outside or in the walls of the container:
  When the combination logic was first created, the position of the new cat to spawn in was determined by the average position of the 2 cats being combined. This sometimes caused the new cat to spawn outside or within the walls of the container. The logic implemented to fix the container was to clamp the spawn position to be the boundary of the container + the radius of the cat.



- Player character not following the mouse position correctly:
  The player movement was based on the mouse coordinates input rather than its coordinates relative to the screen. Once the mouse position was converted to Unity's world position, player movement was as expected.
  Screenshot: N/A because when taking a screenshot the cursor is erased hidden + video required
- Game end not triggering when game end conditions met:
  There were very rare cases where this game would not end when there was a cat above the container due to the game over detection originally being based on a Unity edge collider. Since the edge trigger line is so thin, a cat could sometimes be stacked above another at the game over line and avoid detection. Game end could also be not detected when a cat's high velocity allowed it to phase through the line, since its position was not updated fast enough. By replacing the edge collider with a box collider, the detection range would effectively be increased and eliminate this problem.



  ^The game should end here as the purple cat stacked on the pink cat is overflowing the container
- Spawning of 2 cats held by the player when the previously dropped cat collided with 2 different surfaces at the same time:
  Since the next cat to drop spawns after the one preceding it has collided with any surface and at the end of the frame deletes the script with this logic, the preceding cat may collide with 2 or more surfaces in the same frame thereby calling the SpawnCat function twice before the script for detecting the first collision is destroyed. This was resolved by adding a flag condition to check if a collision had already occurred.

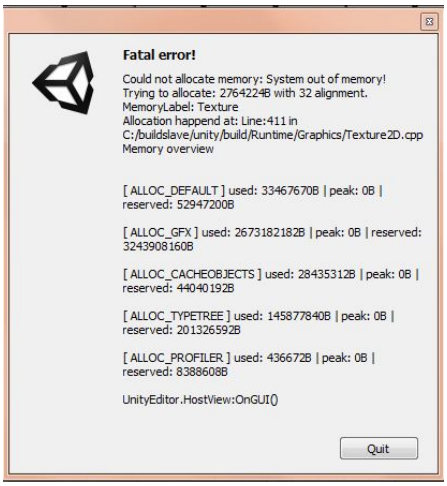


Runtime

There were only two notable runtime errors that were discovered during development and these were both caused by placing an unnecessarily large strain on the processing hardware, causing an eventual crash. They have been provided below:

Three or more cats of the same kind colliding at the same time generated an infinite loop of 3 cats of the next tier being created and combined each consecutive frame until Unity crashed, which was nearly instant every time. Sometimes this would display the error message “ FATAL ERROR: Could not allocate memory: system out of memory” Other times no error is given but Unity is crashed and had to be shut down using task manager. This error could be triggered before the collision logic was handled by a singleton as a result of multiple combinations being processed in the same frame, with the positions of the cats not being updated continuously.

Screenshot of what the fatal error would have looked like because I can’t replicate this as the original code is long gone and replaced with the singleton:



Or watch this clip which is the exact same thing as what happened to me: https://youtu.be/QbwgQSwMSGM?si=6pF070oLYqvImbLM&t=241

The final runtime error does not cause a crash immediately and is quite telegraphed as the game performance noticeably deteriorates as more cats are dropped into the container. Before the colliders of the cats were redone manually, the colliders were automatically generated by Unity based on the sprite, but they would be resource intensive due to the high amount of edges they contained. When Unity updates the physics simulation, the contact calculations required for cats in every frame were very high due to the high edge count causing lag with each further generated cat exacerbating the issue until the game freezes or crashes. This error was logged on 20/1/24.

Screenshot: N/A because game just crashes and the Unity editor is closed as a result

Part B

Efficiency and elegancy of code

Optimisation of performance

Although performance is not a concern in my game due to its low operation requirements, it is still a consideration made for the sake of maintainability and scalability.

In the following script which is attached to every cat prefab, the only way to implement collision detection in Unity is to check every frame whether it has collided then setting a boolean to true when it has. However, this leads to the checks still being run every frame after the first collision has been detected, so the instance destroys itself using “Destroy(this)” after the first collision to reduce the load on.

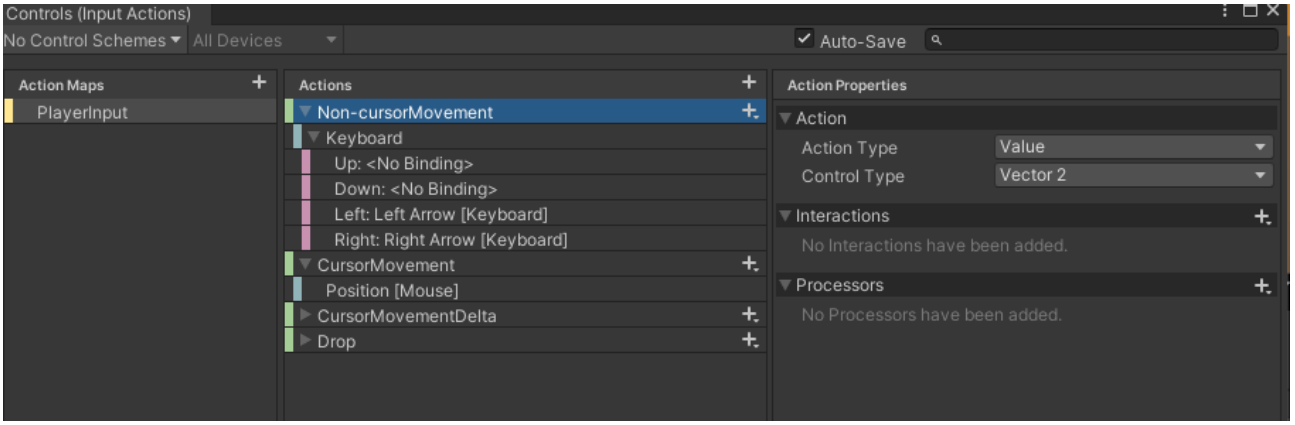
```
public class ColliderInformer : MonoBehaviour
{
    public bool hasCollided = false;

    void OnCollisionEnter2D(Collision2D collision)
    {
        if (!hasCollided && !triggeredFromCatCombine)
        //condition check necessary or else causes bug where 2 cats are spawned if current cat collides with 2 objects in the same frame or from combine collision
        {
            hasCollided = true;
            PlayerMovement.instance.canDrop = true; //enable spawning again
            CatGenerator.instance.SpawnCat(); //spawn next cat when cat has collided
            Destroy(this); //destroy this script as no longer needed and to improve performance
        }
    }
}
```

Adaptability to future input changes

Rather than reading each input individually and adding a new line of code each time for an additional input key by using functions such as Unity’s Input.GetKeyDown(KeyCode.LeftArrow), I

have opted to use Unity's new input action system, which allows for multiple input bindings from various devices to be mapped as an "action" in the following menu.



This allows me to simplify my code for player movement to just be written as "ActionName.action.ReadValue" which accommodates for any future changes to input types/keys without having to add/rewrite the logic of the code since it just has to be added to the input action map.



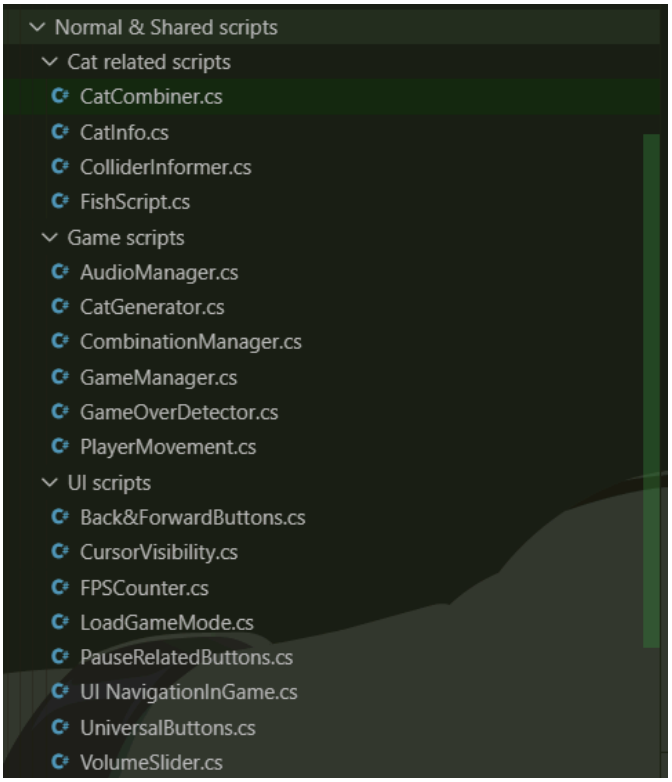
```
1 reference
void UpdateMovement()
{
    //movement logic. If mouse is being moved then only mouse input is used otherwise keyboard input is used
    Vector2 cursorChange = cursorMovementDelta.action.ReadValue<Vector2>();
    if (cursorChange.magnitude > 0)
    {
        Vector2 mousePosition = cursorMovement.action.ReadValue<Vector2>();
        //convert position by mouse to it being relative to camera
        Vector3 worldMousePosition = Cam.ScreenToWorldPoint(position: new Vector3(x: mousePosition.x, y: mousePosition.y, z: 0));
        newPosition = new Vector2(x: worldMousePosition.x, y: transform.position.y);
        /*
        Debugging used for finding position
        Debug.Log(worldMousePosition);
        */
    }
    else
    {
        //movement input from non-cursor devices such as keyboards
        Vector3 noncursorMovementInput = noncursorMovement.action.ReadValue<Vector2>() * Time.deltaTime * moveSpeed;
        newPosition = transform.position + noncursorMovementInput;
        //Debugging used for finding position
        //Debug.Log("reading noncursor: " + noncursorMovementInput);

    }
    //restrict player movement to within the box
    newPosition.x = Mathf.Clamp(value: newPosition.x, min: leftBound, max: rightBound);
    transform.position = newPosition;
}
```

Modularity

Since my logic is written using C# which is an object-oriented programming language, code can be easily reused to reduce the need for rewriting it. This reduced the amount of code required for

creating the other game modes in my game wherever possible to reduce boilerplate code. Instead of rewriting another script for the FPS counter for example in Explosive mode, I just created a new instance of it for that scene. Below are the files used for my base game mode and sometime shared across the others. With functions and objects separated into individual files when appropriate, files become inherently easier to reuse and organised, thus making the program more elegantly structured.



Reducing clutter in module mainline

Rather than stuffing all the logic within the mainline of a module and making it long, the code has been split into subroutines with names that indicate their purpose, allowing for the code to be easier to follow. This has the additional benefit of making the logic reusable for other purposes and furthers the modular design of the program's architecture.

```
void Update()
{
    if(!GameManager.instance.gameOver&&!PauseRelatedButtons.instance.Paused)
    {
        UpdateMovement();
        CheckDropAbility();
    }
}
```

Refine the user interface Final UI design (screenshot)

Main Menu

The grey cat on the left is animated, so the tail position is different every time I take a screenshot.



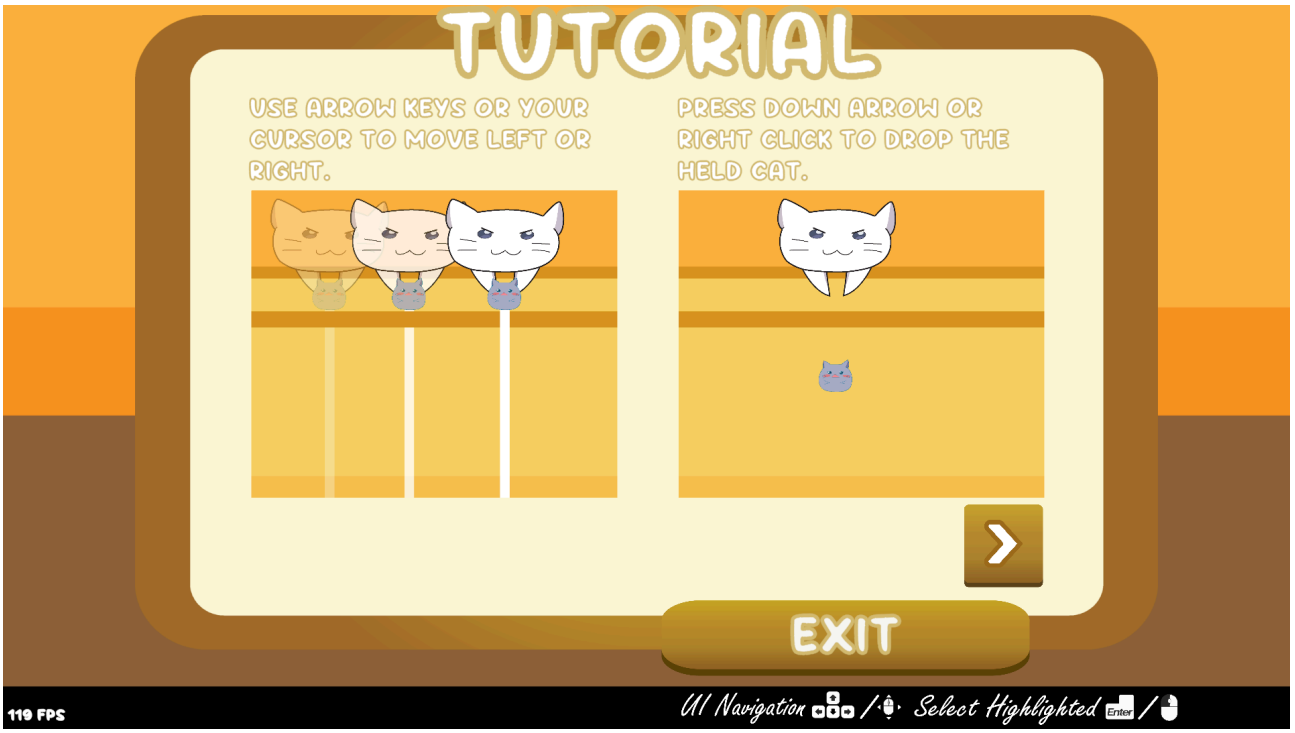
Game Mode Selection Menu



Settings Menu



Tutorial

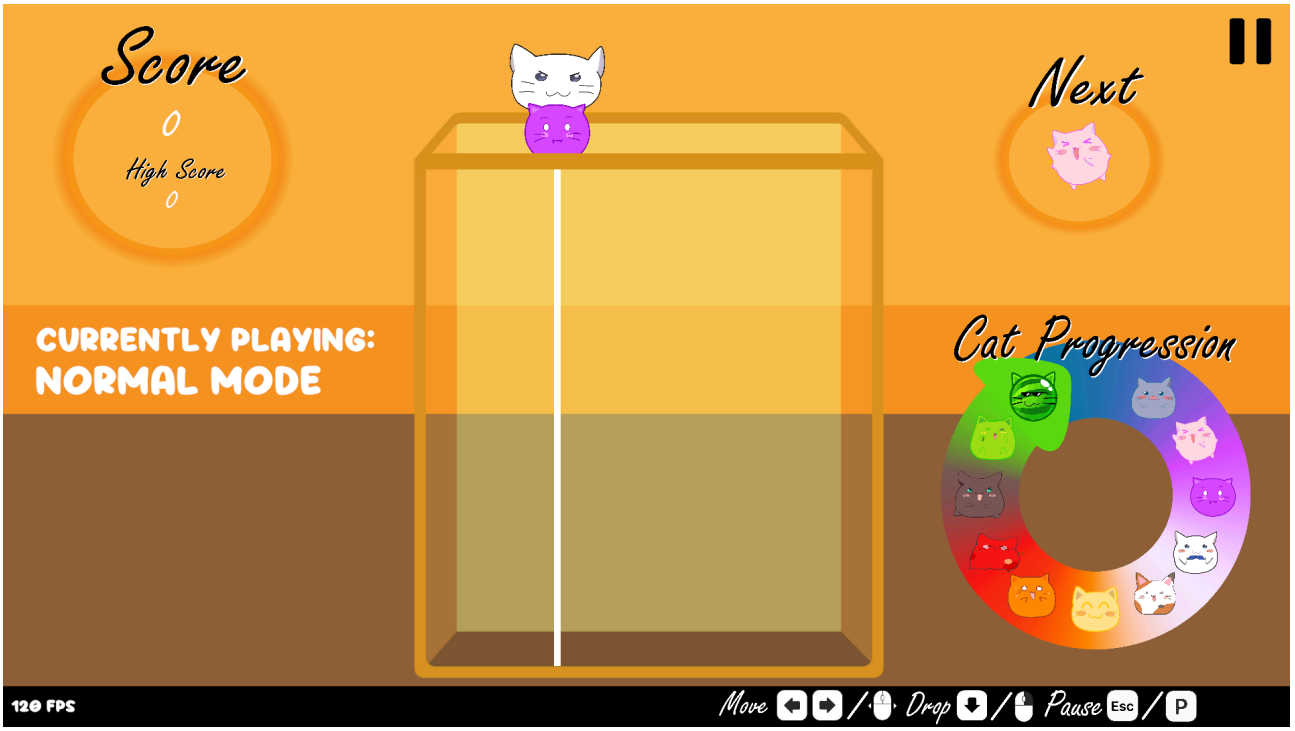


Credits Page



Screen in a Game Mode

The text at the middle left displays which game mode is currently being played.



Pause Menu



Game Over

The new high score text is animated with a glistening effect and only there if a new high score is achieved for the given game mode. The middle has a screenshot of the most recent game state.



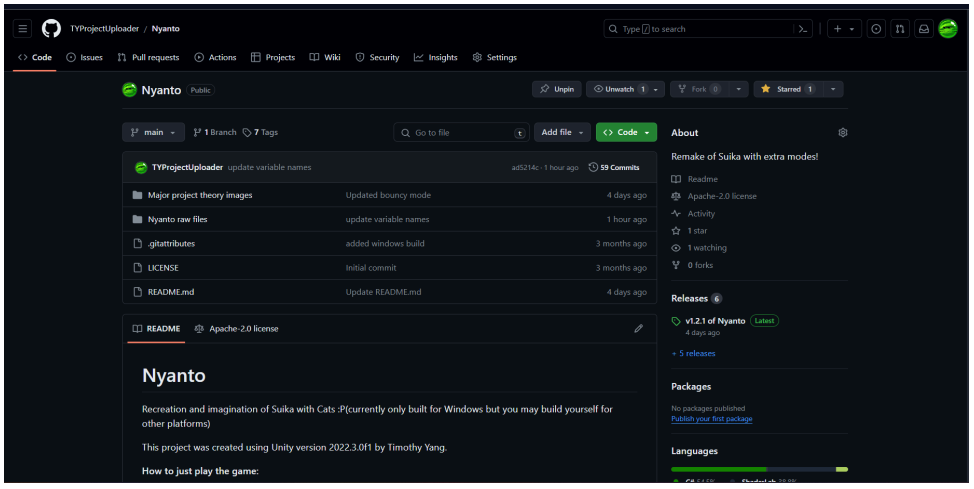
CASE tools to monitor changes and version control

recommend you identify CASE tools across different phases of the project management cycle?
There are various CASE tools for monitoring changes but they can be split into two main categories. Distributed version control system (DVCS) or Centralised Version Control System (CVCS). Both use a repository system where version histories are stored to allow changes to be reverted easily. The former provides each developer a local copy of the repository, allowing them to create branches and experiment locally as well as resolve conflicts locally before pushing to the central authority. In the latter, there is a single central repository that serves as the authoritative source for the project's version history which limits many version control operations such as committing, merging, and history viewing without connecting to the central repository.

Due to my game being a one-person project, the choice of the system is not particularly important. But if the project was to expand to have multiple developers, opting for DVCS would always be more optimal as DVCS provides more flexibility, enabling multiple people to work independently and offline since there's no need for a central server. This is because changes are merged locally before pushing them to the central repository, reducing the likelihood of a conflict on the main repository, as conflicts can be first caught and resolved locally. CVCS' main issue would be that it has only one single copy of the entire repository, with developers only having limited historical versions on their PC. DVCS has the advantage of having entire copies on each collaborator's PC, allowing for greater flexibility with development as changes can be committed locally and synchronised later. This ability to commit changes to a local repository allows for faster local operations.

Git is an open-source DVCS CASE tool I have chosen for my project as it is the most popular with nearly 90% of the total market share. Git's version control allows developers to trace back versions to find a bug in addition to logging changes. Git also enables source files and change history to be uploaded to GitHub for easy sharing and as a cloud backup on top of being a way to have a central repository where changes can be approved and logged if I were to ever add more developers. GitHub is a code-hosting platform that provides an interface for Git's version control and change monitoring. Due to git and GitHub's popularity, it would also mean documentation and online support would be extensively available as are third-party services. Other potential DVCS could have been Mercurial which is simpler in terms of features or Helix Core (costs money for teams of 5+) which has extensive access controls and can operate as either a DVCS or CVCS. However, due to these CASE tools lacking popularity, they would have fewer support resources and people with the expertise required for them if I were to try to collaborate on this project.

Nyanto's GitHub page is: https://github.com/TYProjectUploader/Nyanto



Catering for possible changing user requirements

To increase the ease of catering to any change to any files, each file in the unity project has been appropriately labelled and grouped with folders so they can be found easily. The main purposes for changes to user requirements would be to update gameplay mechanics, fix bugs, change in game UI, and increase platform/device support. To enable future changes, all source assets and code will be kept along with templates for asset creation (e.g. tutorial board layouts). The source files can then be opened with Unity or other required software for editing and the creation of assets and features.

Once the change required is identified, the module that needs to be updated or created can be located by viewing the structure chart that will be included as part of the GitHub repository. By including the structure chart, developers other that may have not originally worked on the project can quickly understand the system's architecture. The game has also been made with a modular architecture, allowing for easy addition, removal, or modification of features without disrupting the core game play.

For supporting possible changing platform support/device in the future, the logic architecture for control and navigation of the game has been made with Unity input actions to allow keys to be effortlessly remapped. Additionally, Unity itself can compile the game for a variety of platforms.

Changes will be made by mirroring a copy of the repository to the developer's local drive, where they can test and experiment with the new changes locally before making a pull request with a description of their changes to GitHub. When all pull requests have been approved after being cross-checked with the original user-specifications, a new version release will be created on GitHub for download. If anything major breaks in the new version, it can always be quickly rewound before the patch with Git.

The major core strategy to cope with changing user requirements would be implementing agile development methodologies. This allows changes to be made quickly with each iteration adding features in small increments or fixing bugs based on user feedback, which will mainly be provided by GitHub comments on the public repository or in-person with play testers.

It should be noted that the repository in public on GitHub and under the Apache 2.0 licence. Any users is free to create or modify the game to suit their personal requirements and contribute to longevity, improvement, and expansion of the game. Even without further development support by the original developer (me), the game can be improved indefinitely while the GitHub repository persists on the cloud.

User manual



About Nyanto:
Nyanto is a casual puzzle game where players drop cats into a container while trying not to overflow the container and gain as many points as they can. When two identical cats are stacked upon one another, they combine into a cat of the next tier and grant points. Combinations can be chained together in a short time period to multiply the score that would normally be gained.
This is an open source project meant to further the innovations within the Suika-like genre.

Download the latest version here: https://github.com/TYProjectUploader/Nyanto/releases

Getting started (after following the installation manual)
Menus can be navigated by mouse or selecting buttons using the arrow keys!



Check out the tutorial for an in-depth guide to playing the game, or just start reading from "How to play" for a summarised version.



Settings
There's an option to turn off screenshake for explosive mode. You can also change the volume from the pause menu while in a game mode at any time.

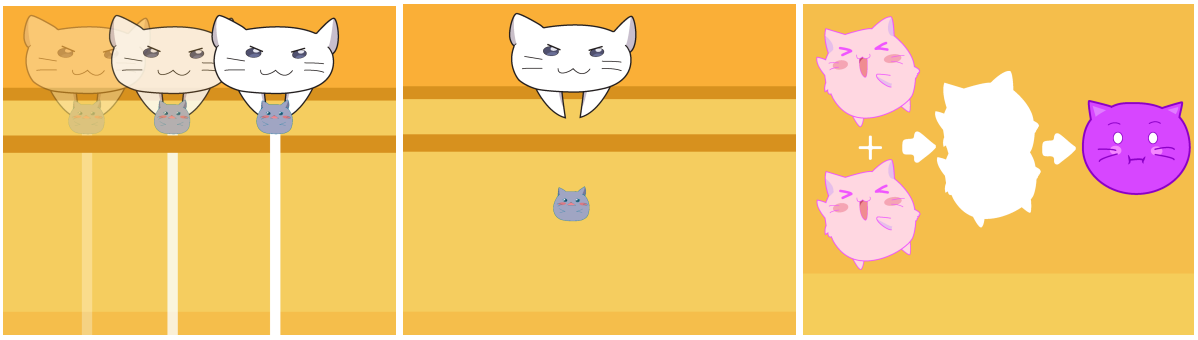
Game modes
There are 3 game modes to choose from after clicking play.

Standard: the base game with nothing funky
Bouncy: the cats bounce around in the container
Explosive: the cats explode when they combine



How to play

Use the left/right arrow or your cursor to move left/right and the down arrow or right click to drop the cat you're holding. Merge cats of the same tier together to score points.



How to win

There is no winning in Nyanto, only bragging rights for getting a watermelon cat. It's like Tetris, you can keep playing to combine cats until your container overflows or a cat flies out of the box.



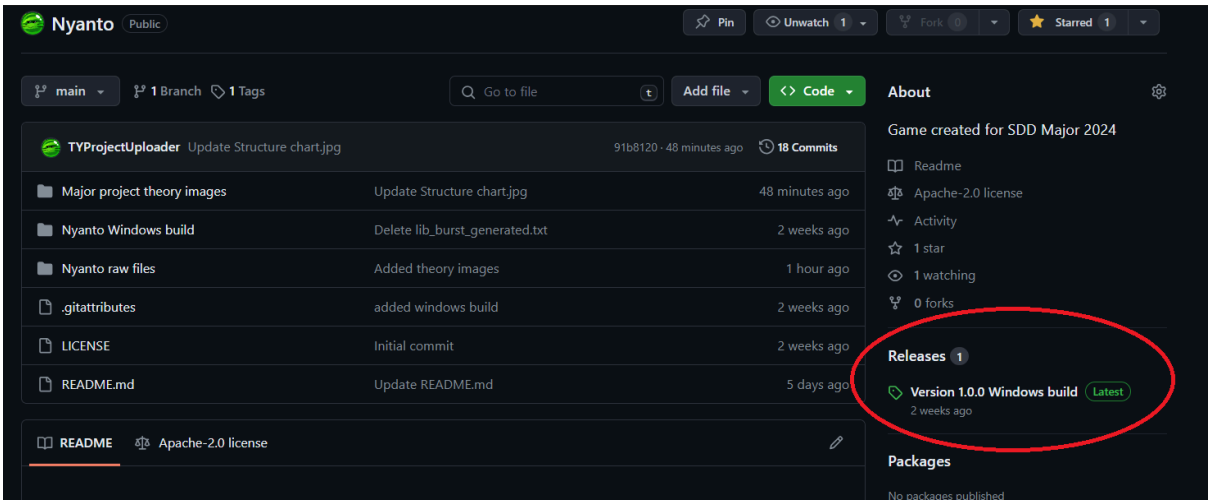
Challenge your friends to see who can get the highest score or to get a watermelon cat! (VERY DIFFICULT)

Installation manual (Game ONLY, not source files)

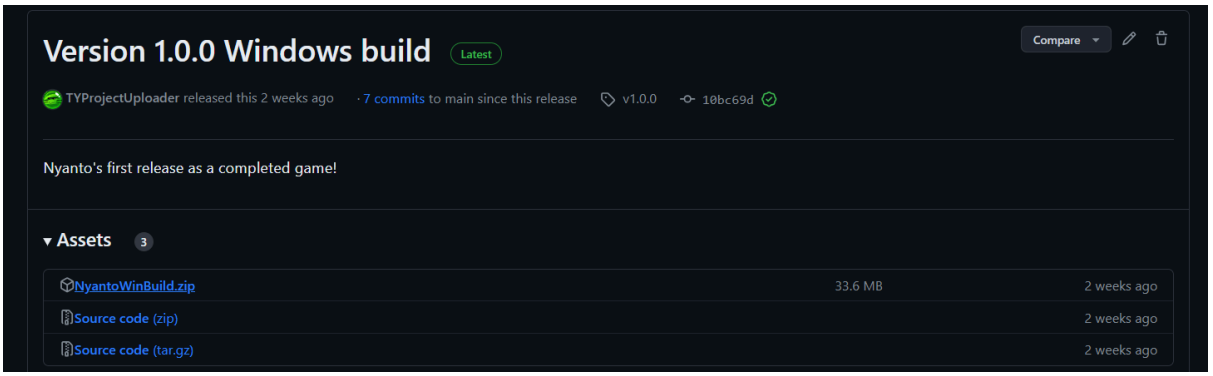
Minimum specs	Recommended specs
OS: Windows 10 or newer, 64-bit Processor: Intel Core i3-6100 / AMD FX-8350 or equivalent GPU: Intel HD graphics 530 Memory: 1 GB RAM Storage: 1 GB available space	OS: Windows 11, 64-bit Processor: Intel Core i7-7700 / AMD Ryzen 5 3600 or better GPU: NVIDIA GeForce 1050 or better Memory: 4 GB RAM or better Storage: 2 GB available space or better

Installation Instructions:

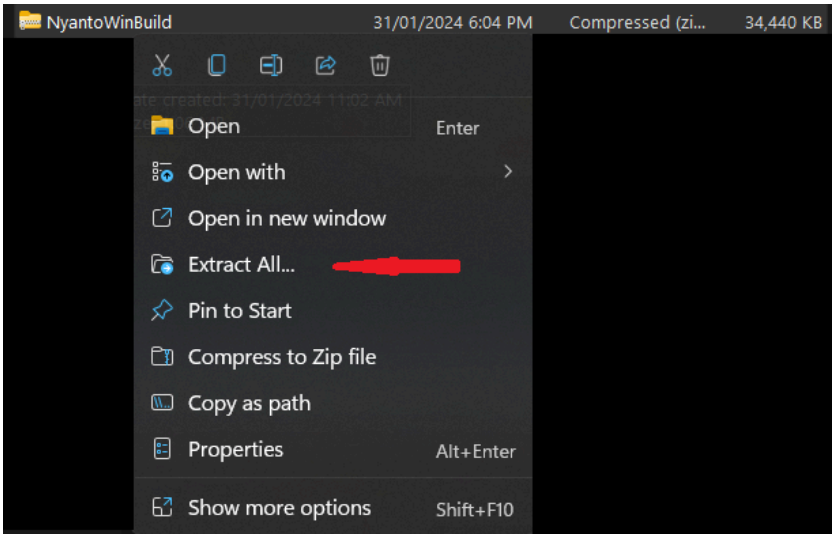
1. Open a web browser
2. Enter this link: https://github.com/TYProjectUploader/Nyanto
3. Click on the latest release under "Releases"



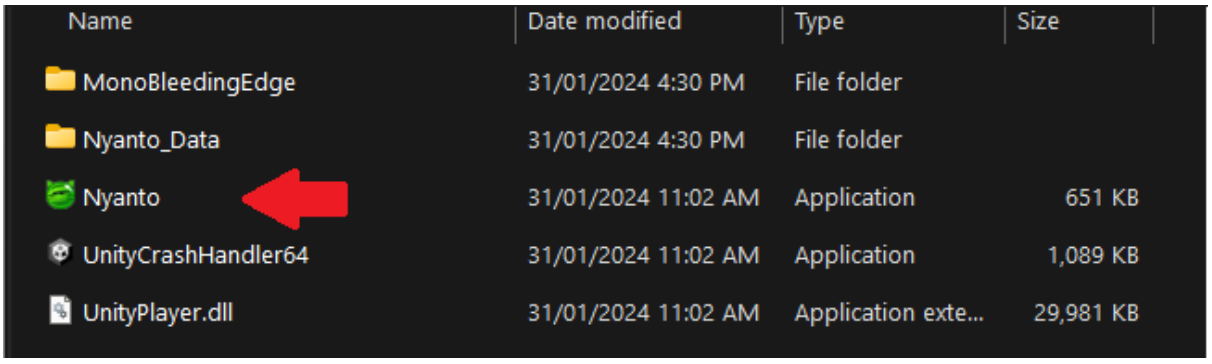
4. Double-click on the file titled NyantoWinBuild.zip to download it



5. Right-click on the zip file and select “extract all” to download the game files to your chosen location. (You may delete the original zip file afterwards)



6. Open the new folder created and select the file named Nyanto to launch the game



Digital copy of all source code

All source code including the various files used to create this project can be found at https://github.com/TYProjectUploader/Nyanto

Project report

What I've learnt

Despite already having dabbled into using Unity for a 2D game in a previous project, I was unprepared for the time sink that creating Nyanto would be. I had largely forgotten how to operate Unity having not opened it for over half a year and had to relearn basic things such as collision detection by referring back to my previous game. This further demonstrated the importance of keeping records of previous works, which I already do for other subjects as one day they may come in handy. Originally, I had only intended to develop the game over 2 full weeks (8h each day) over the Christmas break. But it came out to more than that due to various unforeseen challenges, expanding scope of requirements or lack of knowledge required for the task I wanted to do.

Greater artistic skill and expertise in illustrator

Making the art for the game was where it all started. Illustrator was my choice due to my existing familiarity with it having taken graphics design as an elective subject previously. The cat drawings were fairly simple with each taking 0.5-1h with each progressive one taking less time as my skill improved. However, the utilisation of gradient for UI elements was uncharted territory for me, as was dealing with font and special effects for text. I sought out help from YouTube tutorials and quickly picked up the necessary skills, which I can apply for future art projects.

Singleton design

A singleton is a design pattern that ensures a class has only one instance and provides a global point of access to that instance, making referencing a certain script easy. I came across this when trying to figure out ways to prevent the infinite combination bug present in my game which had many logic flaws but were fixed by having one singleton run the logic required for combining all cats. Singletons were used for both audio management and combination management in my game. Now having unlocked this too, using OOP has become more intuitive for me.

Unity development

In the early versions of the game, often things were breaking for no apparent reason to me. I had checked my code multiple times and found the error was not with me but rather with Unity after seeking some online forums. Unity projects have various settings that allow you to tweak how the game is run. For example, the amount of physics calculations could be increased or limited, the highest velocity of an object could be limited, when an object is still considered to be touching another and so on. By tweaking these values I was able to fix many unexpected behaviours I was getting like things clipping into each other and am confident I now have the skill to competently develop a 2D physics based game in Unity again.

Usage of git

I had never used git before I started this project. Over the course of working on the project I was forced to learn the basics of using GitHub, git and git LFS. There is no doubt in my mind that these skills will give me a head start when learning comp sci in Uni.

Debugging in VS Code

It took me about an afternoon just to understand how to set up the Unity debugger. The guides I found were out of date and using 3rd party tools as they were released before the official Unity plugin had the feature available. I had no clue how easy it was to implement and use before I got it working.

Importance of other perspectives
When handing off my game to play testers, I was generally confident that no major game breaking bugs could be found, but boy was I wrong… There was one particular playtested who did unexpected things that I never considered during my own testing, such as spamming pauses while explosions were going off in explosive mode or finding gaps in my game end detection. Without seeking people who may not have approached my game conventionally, I may have never had those bugs fixed.

Importance of spamming Ctrl s in Unity
Unity doesn't have auto-save as a feature, I lost an hour's worth of work when it crashed. :/

Future directions

There are various ideas that were suggested to me by play testers that seemed promising and interesting. The two major ones would likely be multiplayer and ability to assign custom key binds. Other ideas floated were new types of game modes and greater varieties of special items that can be dropped (like the fish and bomb). The features that have been proposed would've been difficult if not impossible to implement in the given time and technical knowledge restraints, but are definitely worth looking into if I ever get free time.

Outside the feedback I've been given, there are various things I'm not completely satisfied with but lack a solution for at the moment. Notably, the UI, colour design and audio of the game doesn't feel like it would be something that could ever be featured on Steam. Animations could be added to the cats in the container and when inside a game mode the bottom left screen could be filled with something like a global leaderboard. With enough time investment I could improve my graphic design skills, intuition for picking colour schemes and coding skills to implement the aforementioned issues. Another possibility would be to commission professionals to provide the assets I required for the things I envision, as it would be more economical.

Currently, Nyanto is only supported for Windows, but it has the architecture in place to allow for porting to other platforms. This is a consideration for the future as it would make it more accessible to users across different platforms and increase my repertoire of development skills.

Revisiting the project once I have gained a greater mastery of object-oriented programming would also allow me to see various inefficiencies that could be fixed to optimise the performance of the game further. But for now it will just remain open source as a public repository on GitHub for anyone to contribute to or interested in making their own Suika-like.

Comparison with original specifications

Developer specs were mostly all met, with few that were partially-met. Any partially-met specifications were decisions made that were better than the original plan laid out by the specification. Overall, following the dev specs was very successful.

Specification	End result & discussion
The overall development will loosely follow a structured approach and switch to an agile approach to account for any minute changes until the submission deadline.	Met The core game with its game modes were created quickly during the Christmas break and uploaded as v1.0.0 to GitHub. Since then, multiple minor tweaks have been done to enhance the experience of the game based off play tester feedback using the agile approach.
Utilise copyright-free audio files for sound effects and background music and tune using Audacity.	Met All files used for the audio have been used within the bounds of the free licence provided and credited in the bibliography and on the GitHub repository.
Create illustrations required for the game using Adobe Illustrator	Partially-met Some illustrations (the explosion and input prompt icons) would have taken me too long to implement myself and I would not have been able to draw them as well, so they were taken from the internet and correctly credited as well.
Game development conducted using the Unity game engine and C#	Met Nothing to add.
Code formatting is consistent with C# naming conventions such as Pascal for functions/classes and camel case for variables.	Met Although this wasn't met during the core development of the code, code has been cleaned up since.
Design code for reuse and modularity	Met Code has been done as such and was greatly helpful towards building the other game modes as it reduced the workload.
Code is internally documented with comments to assist other developers in understanding the logic as an open-source project	Met Code is open source and all on GitHub, with annotations throughout!
Code generates logs of debugging statements for major modules	Partially-met When the project is opened in the Unity Editor, debugging statements accompany many major modules. However, many have since been commented out to reduce the clutter of random information but still exist within the code so they can be uncommented out at any time.

The entire system should be mapped out with a structure chart	Partially-met Only the base game mode was mapped out, but this was a compromise based on the fact the rest are similar in their structure and repeating the structure chart 2 more times would make things confusing.
Handle version control with Git & GitHub	Met Feel free to check it out on GitHub!
Select play testers from a variety of people to test quality throughout development with WebGL builds and survey for errors/ergonomic issues	Met Roughly 15 people have tested the game at various stages, all with different levels of game playing experience and familiarity with Suika-likes. Their feedback has been valuable in improving the UI and physics of the game.
Design algorithm architecture to accommodate platform porting with Unity Input action system	Met Just a few clicks and no extra lines of code allow the game to support all kinds of devices!

User specifications were nearly all fully met. It is only held back by subjective opinions, which unfortunately didn't get solutions developed that rectified the issues discussed. In culmination, following the specs has been deemed as a success.

Specification	End result & discussion
UI is navigatable by mouse and keyboard in an ergonomic way and intuitive.	Partially-met Some play-testers found that using right click to drop was unintuitive as they preferred left click while others found it fine. Due to a non-conclusive consensus on which one was more preferred, it has remained right-click and makes the specification only partially-met. In terms of UI being navigable by mouse or keyboard only, it has been completely successful.
Buttons of the next page are defaulted when using keyboard to navigate the UI.	Met This has assisted in making the experience more intuitive and ergonomic.
UI and screen design use standardised icons and has consistent design (including font) that is intuitive.	Met No complaints received by play-testers.
Appropriate and non-intrusive audio feedback for game events and UI usage.	Debatable Input prompts were non-intrusive and easy to reference at any time. However, explosive mode features a fairly loud explosion when bombs are combined that may be considered intrusive for people playing with headphones. Arguably this is appropriate for the context and non-intrusive as sfx

	can always be turned off.
Colour of UI elements used complement each other and cause no visibility issues.	Met The colour scheme is a consistent amber/orange/brown tone. Previously, button colours were too similar, making it hard to know which was selected, but this has since been rectified.
Game has clear and simple in game tutorial.	Met Short, simple and sweet! Check it out by clicking tutorial in the main menu.
Compatibility with modern computers that have Windows 10/11 installed.	Met Tested on both! No issues in either.
Resolution of screen suitable for 16:10 and 16:9 aspect ratios	Met Tested on both! No issues in either.
Appropriate messages or prompts (neutral tone) given for navigation or notable game events	Met Most messages were short and had no way to be expressive.
Game can be paused with keyboard shortcuts “p” or “esc”	Met People were pausing the game with ESC without evening knowing it was a key bind! This proved that following conventions in games are vital.
Respects user privacy and security and asks if necessary for data collection	Met No data is asked for or stored. So met by not having it be a consideration required.
Game files are below 2GB in size and require no special hardware	Met Less than 500mb are used!
While running the game should not take any more than 1gb of RAM	Met During operation, RAM usage hovers anywhere from 150mb-300mb at min and max.
Game does not require an internet connection	Met Nothing to add.
Smooth running when on at least 30fps with no freezing, long response times or crashing issues	Met When on low fps, there are no issues that are listed. However, physics calculations by Unity may fail or do weird things that they aren’t meant to. Since this wasn’t listed in the original specs, it has been marked as met.

Logbook

Date	Project component	Progress made & issues encountered	Next session to-do list
21/12/23	Game Assets	Started illustrations on cats and background elements using Illustrator.	Start creating the docs for the theory components
28/12/23	Theory stuff, Game assets	Created Google Docs and laid out tasks to be made. Also completed all drawings for the cats with some illustrator help on yt for specific things I wanted to do like gradients.	Create background and radial wheel
29/12/23	Game assets	Created a radial wheel gradient, which I didn't know how to do before using a guide I found on YouTube.	Finish background illustration
1/1/24	Game assets, Game UI	Created background for the game. Ran into issues trying to get the score text and score bubble aligned for various resolutions. Solved by making the score text a child of the score bubble.	Start on social ethical considerations
2/1/24	Theory stuff	Completed social and ethical considerations.	Start creating the base game mode
3/1/24	Creating the box for the cats	Trying to figure out why a certain sprite keeps falling in the simulation, solution was add rigid body 2D component and simulating it as static.	Continue developing base game mode
5/1/24	Player movement	Could not figure out why a tiny mouse movement causes the player to fly off-screen, will try again tomorrow.	Continue developing base game mode
8/1/24	Player movement	After many hours of searching, finally got it to work by using the .screentoworldpoint Unity function and had to create some logic so that only one input could be done at the time. Also did a bit of tidying up so further controls could be added.	Begin developing cat logic and UI
11/1/24	Collision boundaries and cats	Cat prefabs were developed in this session. Spent a few hours trying to fix the issue of cats phasing through each other but hitting other colliders such as the walls fine. Turned out composite colliders with geometry type set to outlines would cause the issue as it uses primitive unity physics. Changing it to polygons fixes this issue.	Create logic for dropping cats
15/1/24	Dropping	Discovered a bug with spawning 2 or more	Create logic for

	cats logic	cats in the held position of the player when a cat being dropped collides with 2 or more objects in the same frame. Fixed by adding a boolean check to see if a collision has occurred already and setting flag to true when a collision occurs. Also completed the next bubble UI and its logic in this session.	merging cats
16/1/24	Combining cats logic	Had runtime issues multiple times while play testing when things combined. Still not sure how to go about fixing the issue as under regular circumstances it works fine. 1AM update: after some tedious desk checking and consultation with Oscar found the issue. It was caused by multiple cats of the identical kind colliding. Fixed by creating a singleton script to handle collisions.	Finish up UI and start on combo mechanic
19/1/24	Combo & score calculation logic & gameover logic + added fish	Finished all without much issue. Starting on illustrating game over UI elements by creating them but still unfinished. I also distributed a few copies of the unfinished version to friend for testing.	Create the end game trigger and end game screen Also add buttons + volume slider and their logic
20/1/24	Cat collliders + screenshot at gameover	Had multiple performance issues across the playtesters I asked for help as the load on the physics engine was too great by using generated polygon colliders. Decided to make the colliders myself and see if it fixes the issue. Result: highly increased performance + 30ish fps. Fish likelihood was found to be also way too high by playstesters. Values have been changed. Screenshot on Gameover not showing correctly fixed by updating the asset folder and waiting for a frame.	Further bug-fixing and playtesting
21/1/24	Bug fixing + mechanics adjustment to dropping	Had a bug where cats would clip out of the top of the box after being discovered by playtesters spamming at the top. Optimised more code. Fixed the bug and changed the drop cooldown to have a minimum value.	Create a pause menu
22/1/24	Game UI	Created pause UI and animations and finished game over UI. Also slightly edited the game over timer and game over boundary location to make the game feel more smooth.	Redo combination logic as there are bugs atm.

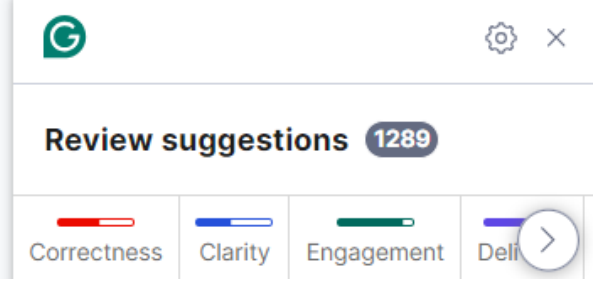
		Some buttons logic/UI still have to be made such as volume sliders and link to home screen (unmade).	Also start on adding sound to the game
26/1/24	Cat combining logic + Sound effects + volume sliders	Completely revamped the combination logic. Now everything is computed by the singleton combination manager and there is a 0.07s delay before the combination occurs to allow the cats to flash white among other changes. This has reduced the load when multiple combinations happen at once improving framerate and also code efficiency. Sound effects + bg music were added with no issue.	Make UI navigable by keyboard/mouse
27/1/24	UI navigation + player controls	Made it so UI could be navigated by keyboard on the main level. Took a long time and the effort was probably not worth it for the results. Also changed player controls so that drop can be done with down arrow for one-hand play instead of the spacebar.	Create the main menu
28/1/24	Start menu, UI navigation, Start Cat animation	Created the start menu and made it navigatable with keyboard and also animated the tail of the cat on the start page. Nothing noteworthy other than the excessive time it took navigation logic. Also changed highlighted/selected button logic to be sprite change instead of animation.	Create the tutorial accessed from menu
29/1/24	Frame rate counter, Tutorial	Spent a solid half hour trying to figure out why [serializefield] image wasn't showing in the inspector but then realised I didn't import the unity UI library. Drew some new buttons too for the tutorial page. Took too long with the tutorial navigation logic and will need to make tutorial images tomorrow. The game's frame rate is now set to update based on the v-sync refresh rate of the end user.	Continue refining tutorial
30/1/24	Tutorial & UI logic	Tutorial is now fully completed. Fixed some bugs with the UI navigation logic and completed more illustrations for the tutorial so now it's pretty much ready for shipping as is. Gave out to more playtesters as well. No issues were reported from playtesters.	Make first windows build with mostly all completed elements
31/1/24	Bugfixing + Windows Defender check	Fixed a bug with the background elements not showing on the main menu when exiting the tutorial or credit page. Also finally added the project to github to allow better version	Start considering working on theory

		control for future updates. By submitting the program to Windows for malware scanning, the error that pops up for an unknown source file no longer shows when downloading the game and playing for the first time.	
1/2/24	N/A	It's now 2 playtesters whom find the layout of the pause button weird with resume not being the top button but rather restart. As Nyanto is intended as a rage game, restart had initially been placed at the top. Further player feedback needs to be acquired before changing the layout of UI. Also worked on various theory elements such as OS considerations and a bit of error discussion	Gather more player feedback and check for further bugs
2/2/24	Theory	A lot of editing of previously written stuff and added some more discussion of errors	Editing and Check B) theory & part A) submission theory
3/2/24	Theory	Edited some previously written stuff and sent emails to the teacher to seek further clarification with assignment details.	File types theory work
4/2/24	Theory (file types and data structures)	Spent a while researching each file type in the Windows build generated by Unity and how they worked. A few file types lacked documentation as only Unity knows how they are handled since they were specific to the game engine so they just took a while to research. Also completed ebnf + railway diagram for the if statement.	EBNF and railroad diagrams
5/2/24	EBNF and railroad	Rewrote logic for ebnf and railroad diagrams after finding some minor errors and inefficiencies	Check B) theory 5 module& part A) submission theory
6/2/24	Theory	Just edited various parts and added more detail in various parts. Also noticed that UI issues would arise if the game is loaded on ultra-wide resolution ratios such as 21:9 . Although since this problem is too niche with most games breaking on this kind of resolution anyway and Unity not supporting defaulting to a set resolution depending on native resolution this will have	Check B) theory & part A) submission theory

		and bandaid fix and the game will be hard set to 1980x1080 for the windows version using letterboxing to prevent content stretching like what most other games do.	
11/02/24	Structure chart	Completed the structure chart in draw.io after many hours and uploaded to GitHub too. Due to sheer size, draw.io could not upload to Google Drive before timing out and it had to be saved locally with an xml file.	DFD & IPO
11/02/24	Discussion of language used	Just spent a bit of time on editing pre-existing theory parts and finished discussion of language used.	DFD & IPO
12/2/24	Structure chart	Noticed a missing flag in the structure chart and fixed it. Also redid the outline for screen designs to make it neater. Also due to new guidelines on user and developer specs + problem statement, they will have to be updated/remade. Also ended up remaking Gantt chart in Gantt project. Quite frankly I don't know if this was worth it.	DFD & IPO. Also need to clarify again with teacher about test data
15/2/24	Check A theory	Remade the design specifications again. Also created the DFD for the frame update module	IPO, data dictionary
16/2/24	IPO Chart, formatting for theory doc	Completed IPO chart and formatted the theory document with a table of content and font~	Data dictionary, five functions/modules + desc.
17/2/24	Five modules and function	Wrote the 5 modules and functions	Need to clarify whether Pseudocode + flow chart is for function not module
19/2/24	Editing of pre-existing theory	There was errors in structure chart and dfd. Especially with DFD not returning anything back to the user.	Need to clarify whether Pseudocode + flow chart is for function not module
21/2/24	Further updates to theory section	Minor issues were found in DFD or context diagram and they were rectified. Five modules/functions were also rewritten. Pseudocode completed with no issue.	Pseudocode and flowchart
22/2/24	Combining cats score	Found bug where score would be added if 2 cats were thrown into handle collision but	Finish flowchart

	logic, pseudocode	were of different type/tier. Fixed the logic error by changing when score would be updated. Flow chart is still in the works due to trying to figure out some elements on Draw.io.	
23/2/24	Bugfixing, flowchart	Flowchart is completed. Bug found with the “currently playing” game mode text not being covered by game end screen panel. Fixed by changing hierarchy of elements in Unity editor	Start on check 2 theory
24/2/24	Desk check	Completed desk check for the combinecat module	Continue heck 2 theory
3/3/24	Bugfixing	A bug was found by bug finder “teddykirrito” (friend) where the game would end in bouncy mode when a white cat was spawned due to the game ending collider touching it on initiation. Fixed now by changing size of collider, but weird how it wasn’t caught before.	Fix alignment of sliders Pause menu has white cats when combining
8/3/24	Theory editing	Based on feedback provided for check 1 all issues pointed out have been resolved	Submit part a) Update ebnf statement to have a casewhere statement to demonstrate better understanding?
20/3/24	Adding explosion effects to explosion mode	Adding the explosion visual effect caused this error to pop up about leaks on jobtempalloc. According to online articles this seems to be a long-running internal Unity bug that has not been fixed for years.  Afterwards, a camera shake effect was added.	Update ebnf statement.
21/3/24	Adding epilepsy warning	Due to adding screen shake to the explosive mode, found the game was flashing a bit too much and making myself dizzy. I added a warning for the game mode. There was a bug with the epilepsy warning not being given and freezing the game if explosive mode was re-entered from main menu. This was fixed by changing it so that the warning was only given once per launch of the game by saving it as state in playerprefs.	Update the ebnf statement.

24/3/24	Bug fixing epilepsy warning	Realised if the game was closed using alt +f4 the epilepsy warning would never be given again. This is fixed by resetting the playerpref stat using unity's Onapplicationquit() method	Update ebnf statement.
30/3/24	UI visibility update	Tried experimenting with changing the colour scheme on the buttons to make them more visible when selected and outlining the buttons too. Outlines look wack when the button is stretched out, so I opted for the colour change being more prominent. In the process also accidentally deleted the folder that houses the UI elements, breaking everything for a while :/.	Update ebnf statement. Maybe also add more things to explosive mode. Start cleaning up code annotations
16/4/24	Adding bomb to explosive mode	Replaced the fish in explosive mode with a bomb… Yea, 2 bombs combining = massive boom. No comments other than the fact I'm sitting here wondering why drawing & adding the logic for the bomb took 3h.	Update ebnf statement. Start cleaning up code annotations
17/4/24	EBNF/railroad diagram Rework of physics on bouncy mode	Added the switch statement. Bouncy mode physics have changed so that combined cats have more than elastic collision. This makes the game mode now like a ticking time bomb until its unplayable and makes it much more difficult and interesting.	Add a checkbox to turn on/off screen shake
18/4/24	Screenshake checkbox Explosive mode bugfixing	Fixed bug where the screen would continue shaking after being paused by reworking the way time is calculated. Also, now added option for turning off screen shake.	clean up code annotations
19/4/24	Test data	Created the test data and started testing the boundaries for the CombineCats module. Nothing notable was discovered.	clean up code annotations
21/4/24	Cleaning up code and annotations	Checked variable names were named according to common conventions and replaced those that weren't. Comments were also added/removed as needed.	User Manual needs to be created.
22/4/24	Changing when the explosive warning is given User	A bug was encountered where the screen shake option defaulted to on regardless of prior settings. This was discovered by using debugging output statements and rectified by moving logic from Awake() to Start(). This also led to discovery of bug where the highscore was not being updated.	Start & finish the project report

	Manual	All has been fixed and now uploaded to GitHub.	
23/4/24	Spell checking and formmattin g the document	 Grammarly is something else… I've tried to fix up all the spelling mistakes so that the document is easier read.	Prepare for submission and send out game for final rounds of play testing
24/4/24	Wrapping up the project	Sent out the game to play testers again. If any issues arise then they will be fixed but until then this may be the last log.	
1/5/24	Outlining buttons and reworking buttons entirely	To give further visual clarity to what is being selected, an outline has been added to the buttons. This was meant to be a 10 minute project that ended up taking over 4 hours as I had to redo how buttons were made as my previous way was inefficient. A single prefab for most buttons have been made so future changes to buttons can be done very quickly (WHY DIDN'T I DO THIS AT THE START? IDK). I guess it's a bit better now… maybe?	Consider making transition between keyboard and mouse input more smooth if there is time
4/5/24	Fixing up railroad diagram	Discovered railroad diagram was wrong when doing an optional loop. This has now been rectified.	
10/5/24	Smootheni ng transition between inputs	The function to now use cursor and a submit button (enter) to navigate buttons has been added, which took longer than expected. Doubt this will ever be used, but I guess now it's more accessible?	
12/5/24	Bug fix	Fixed bug where returning to main menu would make you select a button by default due to the position of the cursor.	Wait for check 2 feedback before final submission
23/5/24	Final checks and update to theory	Having received feedback for check 2, the issues that are present have been fixed up and now uploaded ready to be marked.	

Bibliography

Date accessed	Description	Usage	Link
19/12/23	Suika game knock-off	Reference for physics and behaviour of my game	https://suikagame.com
19/12/23	Suika game knock-off but with sea-creatures	Used for reference for physics and behaviour of my game	https://store.steampowered.com/app/2661210/WhaleGameOnline/
19/12/23	YT video going over basic requirements for Suika. Author: Sasquatch B Studios	Followed the general layout of his plan for the game	https://youtu.be/liDKiD6yv8E?si=y3625OxWbtDsg81_
19/12/23	Official unity article on colliders	Figure out how to make colliders of various shapes	https://learn.unity.com/tutorial/applying-2d-colliders-for-physics-interactions#
19/12/23	Video on Suika being made and being played with AI Author: Code bullet	Used to get ideas on how game should be made	https://www.youtube.com/watch?v=QbwgQSwMSGM
20/12/23	Video on how to make radial gradient in Illustrator Author: instagraphics	Figure out how to make radial gradient for assets	https://youtu.be/k5OvmoixlfQ?si=sttmC4-0umTDE8Vx
1/1/24	Icons for input prompts Author: Julio Cacko	Used as asset in my game	https://juliocacko.itch.io/free-input-prompts
1/1/24	Guide on using custom fonts in unity Author: Catodevs	Figure out how to use custom fonts	https://youtu.be/gMd0xDEFE20?si=YOd27_6yZEmIg7MI
5/1/24	Unity documentation on actions	Used to figure out how to implement player controls	https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/ActionBindings.html#disambiguation

5/1/24	Unity tutorial on action input Author: Sunny Valley Studio	This is the backbone of my player movement controls	https://youtu.be/-c_LYQgG8BY?si=YFEjyLbg8B8mOhxq
15/1/24	Reddit post on common issues with colliders/rigidbodies	Used to figure out some clipping issues	https://www.reddit.com/r/Unity3D/comments/1ee65w/having_wonky_collisions_rigidbodies_being_weird/
19/1/24	YT tutorial on setting high scores Author: Brackeys	Used for my game over function	https://youtu.be/vZU51tbgMXk?si=tn9hye-HPeanEUd1
19/1/24	Font package with open font licence Author: Stefie justprince	Font used for some UI elements	https://www.fontspace.com/puffy-font-f95957
19/1/24	Video on reading images from file Author: c00pala	Used to figure out how to store and access the screenshot I'd use on my end game screen	https://youtu.be/lFp8Z_3wa7M?si=TLBUl32Dosi0ZWnF
20/1/24	Video on custom colliders Author: Max O'Didily	When optimizing performance used this to figure out how to make colliders better	https://youtu.be/2kdQFcTC-v4?si=34FXNNHNu22bJZVT
22/1/24	Video on making a shine effect with masking Author: Zombie Walker	Used for Highscore alert text	https://youtu.be/sAw2Xi3wjSI?si=339ch5H9NdqH58oi
22/1/24	Website on how to pause in Unity Author: John French	Used to create the pause logic function of my game.	https://gamedevbeginner.com/the-right-way-to-pause-the-game-in-unity/
26/1/24	Video on how to carry music between scenes Author: Rehope Games	Used to carry music between my scenes.	https://youtu.be/xswEpNpucZQ?si=AxGKjozTPvsQDG1o
26/1/24	Nya sound effect Author: Pixabay	Used as the sound effects when combining cats and when the game ends.	https://pixabay.com/sound-effects/anime-cat-girl-6731/

26/1/24	Music Author: geoffharvey	Used as the background music for the game	https://pixabay.com/music/cartoons-cute-creatures-150622/
26/1/24	Button click sound effect Author: lucadialessandro	Used as button click sound effect in the game	https://pixabay.com/sound-effects/tap-notification-180637/
26/1/24	Video on Unity audio mixer Author: Rehope Games	Used to figure out audio Mixer unity	https://youtu.be/IxHPzrEq1Tc?si=N2ZC3Bu3vdhLTX9F
26/1/24	Video on Unity Audio sliders Author: Rehope Games	Used to figure out how to change the volume of different items in a volume mixer with sliders	https://youtu.be/G-JUp8AMEx0?si=CI0aR0eoVM2od2Gt
27/1//24	Video on unity input system controlling UI Author: Sasquatch B studios	Used as inspiration alongside unity documentation to figure out how to navigate UI using keyboard input.	https://youtu.be/e30i55Hp-to?si=DXrthEUqsuBOxp2t
27/1/24	Video on unity input system controlling UI Author: samyam	Used as inspiration alongside unity documentation to figure out how to navigate UI using keyboard input.	https://youtu.be/Hn804Wgr3KE?si=uSdRMX-GCszca96C
27/1/24	Unity documentation on mouse warping	Used to move the mouse cursor when pause or game over menu active	https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/Mouse.html#
28/1/24	Unity documentation on physics material2D	Used to make bouncy gamemode material.	https://docs.unity3d.com/Manual/class-PhysicsMaterial2D.html
29/1/24	Video on making a frame counter Author: AIA	Used to create fps counter in bottom left	https://youtu.be/xOCScMQIxrU?si=535D_mxJie58kHEV
29/1/24	Unity article on frame rate	Used to troubleshoot frame rate in a build	https://support.unity.com/hc/en-us/articles/360000283043-Target-frame-rate-or-V-Blank-not-set-properly
29/1/24	Unity thread on things not showing in editor	Used for troubleshooting why an image serializefield was not showing in the editor	https://forum.unity.com/threads/solved-variables-not-showing-in-inspector.392563/

30/1/24	Stack overflow thread on preventing an app launching twice	Used to prevent people from launching multiple instances of the game.	https://stackoverflow.com/questions/24534996/prevent-an-app-from-launching-twice#:~:text=If%20you%20go%20to%20the,a%20second%20instance%20from%20starting.
4/2/24	Microsoft documentation on DLL	Used in research about file data structures to complete the corresponding theory part	https://learn.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library
4/2/24	Unity documentation on Async upload pipeline	Used to figure out what the .resS file is used for as there's little documentation on it	https://blog.unity.com/engine-platform/understanding-the-async-upload-pipeline
4/2/24	Article on the JSON text forma Author: Matthew Tyson	Used to understand what json was used for	https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html
11/2/24	Unity documentation on update cycle	Used to figure out the order of structure chart	https://docs.unity3d.com/Manual/ExecutionOrder.html
11/2/24	Guide on setting up Unity's official debugger	Helped me figure out single-line stepping, breakpoints and so on.	https://youtu.be/dmK4YBwKiTo?si=FJF8gAdwLQLaekUs
20/3/24	Explosion visual effects Author: Osama Deep	Used for the explosion in nyanto Bug fix explosion mode warning	https://assetstore.unity.com/packages/2d/textures-materials/2d-flat-explosion-66932
16/4/24	Explosion sound effect Author: Pixabay	Used as explosion sound effect when combining two bombs	https://pixabay.com/sound-effects/explosion-01-6225/
17/4/24	Unity documentation on the rigidbody2D	Used to figure out how to switch material of a cat while the game is running	https://docs.unity3d.com/ScriptReference/Rigidbody2D.html
23/5/24	Unity article on how they store prefabs	Used to understand how to explain my test data	https://blog.unity.com/engine-platform/understanding-unitys-serialization-language-yaml