

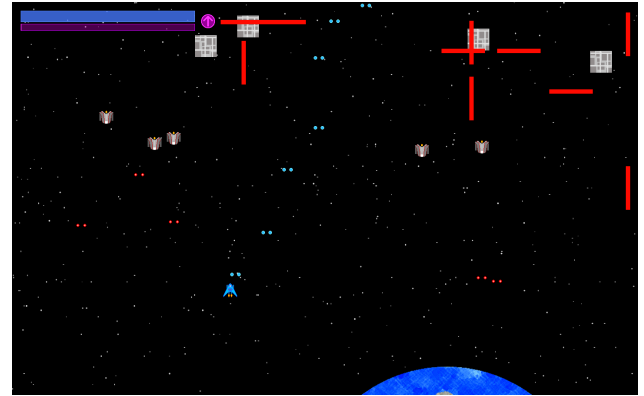
Project #15

SPACE SHOOTER

Design a space shooter game using inheritance

Resources

- In-Class Lectures
- This amazing, in-progress document!



Guides

1. [The Rubric](#) - How your project is graded
2. [Design Notes](#) - How to use this document, and advice about good project design
3. [Enemy Design](#) - Making your enemies unique
4. [First Steps](#) - Making this into a not terrible game
5. [Level System](#) - Divide your game into different levels
6. [Lives and Respawnning](#) - Helping your player to finally get a life
7. [Music & Sound Effects](#) - Adding audio effects to your game
8. [Gamestates](#) - Adding pause and title screens
9. [Upgrade System](#) - Adding a shop system to your game


Future Topics


10. Boss Design
11. Display Effects - Health Bars, Explosions, Ambient Effects
12. High Score Table - File System


The Rubric


Name: _____


How your project is graded


User Interface		25 points	Check	Final
	Title Screen (Must show controls)	5 points		
	Pause Screen	5 points		
	Win / Loss Screen	5 points		
	Health Bar (At least one object in the program)	5 points		
	Dynamic Background (Starfield or similar effect)	5 points		

Player		20 points	Check	Final
	Respawn System (Player has multiple lives)	5 points		
	Respawn Invulnerability (Must be visible to player)	5 points		
	Upgrade System (The player can buy and upgrade equipment)	5 points		
	Functional Upgrades (Two upgrades that <i>change behavior</i>)	5 points		

Enemies		35 points	Check	Final
	Level System (At least six levels)	5 points		
	Basic Enemy Quantity (Requires five non-boss enemies)	10 points		
	Basic Enemy Design (Score based on variance and complexity)	10 points		
	Boss (Has at least one scripted behavior.)	10 points		

Special Stuff - Pick Any Four		20 points	Check	Final
	Music (At least two audio tracks)	5 points		
	Shields (Any visual response to non-lethal damage)	5 points		
	Explosions (Any visual response to object destruction)	5 points		
	Status Effects (Slow, Stun, Fire, etc...)	5 points		
	Ambient Effects (Planets, Black Holes, Asteroids)	5 points		
	Advanced (Rotations, Save/Load, Difficulty Modes, etc...)	5 points		

Penalties		
	Minor Bug	-2 points
	Major Bug	-5 points
	Late Project (any time after start of class on due date)	-10 points / day
	Failure to Follow Turn-In Procedures Properly	-5 points
	Won't Run As Standalone Application	-5 points
	Won't Compile / Run in Processing	-50% of grade

Bonus		
	Your project can earn bonus points for additional special features, high degrees of polish, exceptional presentation, and clever design.	Up to +10 points


Estimated Score: _____ / 100


Final Score: _____ / 100


Senior Rubric

How your project is graded


70 POINTS


User Interface		25 points	Check	Final
	Title Screen (Must show controls)	5 points		
	Pause Screen	5 points		
	Win / Loss Screen	5 points		
	Health Bar (At least one object in the program)	5 points		
	Dynamic Background (Starfield or similar effect)	5 points		

Player		20 points	Check	Final
	Respawn System (Player has multiple lives)	5 points		
	Respawn Invulnerability (Must be visible to player)	5 points		
	Upgrade System (The player can buy and upgrade equipment)	5 points		
	Functional Upgrades (Two upgrades that <i>change behavior</i>)	5 points		

Enemies		35 points	Check	Final
	Level System (At least six levels)	5 points		
	Basic Enemy Quantity (Requires five non-boss enemies)	10 points		
	Basic Enemy Design (Score based on variance and complexity)	10 points		

Special Stuff - Pick Any Two		20 points	Check	Final
				

Penalties		
	Minor Bug	-2 points
	Major Bug	-5 points
	Late Project (any time after start of class on due date)	-10 points / day
	Failure to Follow Turn-In Procedures Properly	-5 points
	Won't Run As Standalone Application	-5 points
	Won't Compile / Run in Processing	-50% of grade

Bonus		
	Your project can earn bonus points for additional special features, high degrees of polish, exceptional presentation, and clever design.	Up to +10 points

Estimated Score: _____ / 70

Final Score: _____ / 70

Design Notes

How to use this document, and advice about good project design

Making The Game Your Own

This document is a guide. It's a series of suggestions. You can deviate from it as much as you like... so long as your results are awesome. Let's consider a few examples of what I mean...

On Naming

💡 Your example code calls a method `spawnEnemies()`. Can I call it `createEnemies()` instead?

✪ Sure. It's your code.

💡 What about calling it `wafflePirates()`?

✪ No. That's bad style. Your name should clearly tell what your method is doing in the code. Even if your game is all about pirates who steal waffles, that method still is not very specific about its role in the program.

On Features

💡 Your example code suggests a strategy that requires players to clear all the enemies to advance. I want my game to be based purely on time. The player needs to survive X seconds on each level. Can I do that?

✪ Yes. If you can figure it out.

💡 Will you give me one-on-one tutoring to debug my alternate approach?

✪ Probably not. I'll give some general advice and answer questions, but I can't sit and debug your code for a long time if you're stuck unless you're struggling with a core project requirement.

💡 How will doing alternate approaches affect my grade?

✪ As long as it isn't some sneaky way to avoid implementing a feature, you won't lose any points. When you want to do something different, ask yourself: is this at least as complex as the suggested route? Generally, alternate approaches will result in bonus points for creativity or challenge.

Enemy Design

Making your enemies unique



The Idea of Design Space

Valve is one of the most highly regarded PC development studios. One of their biggest hits is a team class-based first person shooter called Team Fortress II.

One thing this game does really well is that every class has specific roles and advantages, and none of them are too similar to one another. Everything from their attack ranges and movements to the shapes of their models varies tremendously.

With your enemies, I want you to try and do the same thing. Think of an enemy type as occupying a certain amount of “design space.” That’s to say, if you only have five enemies... you probably want only want one “really fast guy”. The simplest way to do this is think of one unique “signature” thing for each enemy. This might be its movement, attack, or something else. Then, make the rest of it compliment that and just be as different from other enemies (and the player) as it can be.

Archetypes

When brainstorming enemies, it can help to think of common roles or patterns you’ll want in this sort of game. An enemy might occupy two or more archetypes. Here are some suggested archetypes for your game. Things that are harder have an asterisk next to them. There are infinite other ideas, these are just a few....

MOVEMENT	ATTACK	SPECIAL
Moves straight in one direction	Fires one way	Shots cause enemy to be slowed
Really Slow Movement	Fires in a random direction	Shot changes enemy location or speed
Moves Randomly	Fires based on its current speeds	Teleports

Changes Direction Frequently	Fires at the player*	Shoots a bunch of stuff on death
Acceleration	Fires a shot that follows player*	Pushes or pulls the player*
Stays at the top of screen	Fire a shot that accelerates over time	Stealth*
Wraps around screen	Fires a shot that grows over time	
Bounces off edges of screen	Fires a shot that splits into other shots	
Speeds up and slows down	Fires shots in an arc or curve	

First Steps

Player Movement

- Right now the player can only move to the left! Allow the player to move in different directions.
- Make sure player can't move off the edges of the screen

Player Shooting

- Add in a timer to limit the player's shot frequency

Make Evil Squares Cooler

- Evil Squares should shoot. Start simple by having them shoot straight down.
- You'll need to fix the **reactions()** method in **EnemyProjectile** so they disappear on collision. Use the reactions methods in **PlayerProjectile** an example.
- Modify their movement so they move differently. Maybe when they shoot have them change their horizontal direction so they go sideways a little.

Fix Damage System

- Right now, everything in the game DIES INSTANTLY when it takes damage.
- Fix the takeDamage(int amount) method
 - Reduce the GameObject's health by amount
 - Check if it's health is below zero
 - If so, set health to zero and call the die() method
- Give your player a higher health value than 1.

Make It Your Own

- Make the "EvilSquare" have a new name that describes what it is in YOUR game
- Create new graphics for the player and the raider

Level System

Divide your game into different stages

General Plan

Create a variable that tracks the level. Determine when all enemies are defeated, then increase the level. Each time you advance the level, spawn enemies based on the value of level.



A Detailed Implementation

- In **GameEngine...**
 - Create the global variable **level**
 - In your **draw()** method...
 - Call the method **advanceLevel()**
 - In the **advanceLevel()** method...
 - If **stagersClear()**
 - Increase level by one
 - Call the method **spawnEnemies()**
 - In the **stagersClear()** method...
 - Loop through objects
 - If there's an instance of enemy // Hint: use **instanceof**
 - Return false
 - Return true
 - In the **spawnEnemies()** method...
 - If level == 1
 - **spawnRaider(5)** // Your enemy can have any name
 - If level == 2
 - **spawnRaider(8)**
 - Write a method like **spawnRaider(int amount)**, which contains...
 - A for loop that executes amount times
 - Create the object with a line like...


```
objects.add(new Raider(random(width-raider.width), 0);
```

/ Extended Comment */*

The above line makes a raider appear anywhere on the top of the screen. Consider starting enemies at a random level above zero, in order to "stagger" the way they enter. For instance, try setting the second parameter to `random(-150, 0)` instead of 0. How big the negative value should be depends on the enemy speed and how much you want them staggered vs. in a line.

Alternate Ideas

- **Timed Levels** - You could choose to mark special levels as being “timed” and set a timer for them. You’ll need to modify `stagesClear()` such that it checks if it’s a timed level or completion based level, then advances the level when appropriate.
- **Periodic Spawning** - You can make a method that gets called during `draw()` every frame called something like **`spawnEnemiesPeriodic()`**. Inside the method, you’ll have a series of if statements checking the level just like in `spawnEnemies()`. However, only create the enemies every few frames. You can do this using modulus and a timer that simply counts up each frame. For example..

```
if (timer % 20 == 0) spawnRaiders(1);
```

The above line will create a raider every twentieth frame, which is three per second. Probably a little too fast, but you can set the values however you like in your game!

Lives & Respawning

Helping your player to finally get a life

General Plan

Create a variable to store the number of lives for a player's ship. When it takes damage that exceeds its health, it doesn't die - it just resets and decreases lives. The ship only dies when you're out of lives. We will also design a system for temporary invulnerability upon respawn, so you don't die right away again.

A Detailed Implementation

Part A - Life System

- In **player...**
 - Create the class-level variables **lives**
 - In your **constructor...**
 - Set **lives** equal to **PLAYER_LIVES**;
 - In the **die()** method (overrides the method in class `GameObject`)
 - If **lives** is greater than zero
 - Set **x** to **width / 2**
 - Set **y** to **height - 100**
 - Decrement **lives**
 - Otherwise
 - **curHealth** = 0
 - **isAlive** = false

Test Point ✓

Test your project before moving on! Try changing the number of lives - does it still work properly? Now might be a good time to add text displaying your number of lives.

Part B - Invulnerability (First Steps)

- In **gameObject...**
 - Create the class-level variable **invulnerable**
 - In your **constructor...**
 - Set **invulnerable** equal to **false**
- In **player...**
 - Modify the **reactions()** method so that your player only responds to instances of different objects if you are not invulnerable.
 - Remember, if you want invulnerable enemies you'll also need to add code to their reactions methods to make it work for them.

Test Point ✓

Test your project by setting invulnerable to true in the player constructor. Does it work? Once it's working well, you can add a timer & trigger system so it won't last forever, and automatically happen on respawn

Part C - Invulnerability System (Timer & Trigger)

- In **player...**
 - Create the class-level variable **invulnerabilityTimer**
 - In your **constructor...**
 - Set **invulnerabilityTimer** equal to **0**
 - Add to the **act()** method...
 - If **invulnerabilityTimer** is greater than **0**
 - Decrement the **invulnerabilityTimer**
 - Otherwise
 - Set **invulnerable** to false
 - Add to the **die()** method...
 - If **lives > 0**
 - Set **invulnerabilityTimer** equal to **PLAYER_INVULNERABILITY_DURATION**
 - Set **invulnerable** to true

Test Point ✓

Your player should be invulnerable when he respawns. But it's kinda hard to see, isn't it? For now, try printing out the value of the variable invulnerable in text at x, y. This will make it clear when he's invulnerable and when it isn't. In the last section, we'll add one final part to make it display more clearly. Then we can remove the text label.

Part D - Invulnerability System (Blinking Graphic)

- In **player...**
 - In the **draw()** method...
 - If invulnerable
 - if timer modulus 2 equals zero // Makes the player "blink"
 - Draw the player's image
 - Otherwise
 - Draw the player's image

Test Point ✓

Once this works, think about alternate ways of showing the invulnerability. You could alter the image instead of making it blink. Or you could combine the approaches and have it blink between two images (say, your ship and a version of it with a brighter color so it looks like it's "flashing.")

Music and Sound Effects

Adding Audio To Your Game

Making Music

You can find music online in mp3 format, or generate your own using [BeepBox](#).

General Plan

We play music tracks one at a time and loop them. They're rather simple to manage. Sounds are layered on top of each other using different audio channels, so we'll manage these different tracks with an ArrayList.

A Detailed Implementation

Part A - Music

- Big Ideas:
 - We use an AudioPlayer object to play sound files
 - Music loops continuously until we play a new track
 - Before we start one track, we stop the old one
- In `gameEngine`, add the following code:

Code to add to Class GameEngine

```
import ddf.minim.*;

Minim audio;           // declares an object to manage sound
AudioPlayer music;      // declares a player for our music

----- Other Code -----

void setup()
{
  audio = new Minim(this); // initializes the Minim object
  playMusic("spaceship.mp3"); // plays spaceship.mp3
}

----- Other Code -----

void playMusic(String filename)
{
  if(music != null) music.close(); // closes old music, if needed
  music = audio.loadFile(filename, 2048); // loads new file
  music.play(); // plays new file
  music.loop(); // loops the music
}
```

/ Extended Comment */*

Processing uses an audio library called **Minim** to control sound files. Each audio “track” is managed by a single `AudioPlayer`. It has a number of other features and you can explore them here: <http://code.compartmental.net/minim/javadoc/> (Click `AudioPlayer` on the left bottom class menu, then scroll down to method summary).

Part B - Sound Effects

- Big Ideas
 - Sound effects need a different `AudioPlayer` object because we don’t want to stop the music
 - Unlike music, we don’t stop an old sound effect to play a new one
 - That means we need one `AudioPlayer` for each active sound file, so we’ll use an `ArrayList`
- In `gameEngine`, add the following code:

Code to add to Class `GameEngine`

```
ArrayList<AudioPlayer> sfx;                                // Declares the ArrayList

----- Other Code -----

void setup()
{
    sfx = new ArrayList<AudioPlayer>();                    // Initializes the ArrayList
}

----- Other Code -----

void playSound(String filename)
{
    sfx.add(audio.loadFile(filename, 2048));               // adds new sfx file
}

----- Other Code -----

void draw()
{
    for (int x = 0; x < sfx.size(); x++)                   // Loop through sounds
    {
        AudioPlayer a = sfx.get(x);

        if (!a.isPlaying())                               // If it hasn't started yet...
        {
            a.play();                                       // ...play it now
        }

        if (a.length() <= a.position())                    // If it's reached the end...
        {
            a.close();                                     // ...close the file
            sfx.remove(a);                                  // ...remove the AudioPlayer
        }
    }
}
```

- Anywhere you want to play a sound (player shot, enemy death) simply write this line:
 - `playSound("laser.wav");` `// Replace laser.wav with you sound's filename`

Gamestates

Adding a pause and title screen to your game

General Plan

Create a string variable named `gamestate` to track the current mode we're in. Depending on the `gamestate`, call a different method within `draw` to produce different behavior.

A Detailed Implementation

Part A - Pause

- In `GameEngine...`
 - Create the class-level variable `gamestate`
 - In `setup`
 - Set `gamestate` equal to `"gameplay"`;
 - Write the method `drawGameplay()`
 - Cut the contents of `draw` and paste into this method
 - If the user presses 'p' set `gamestate` equal to `"pause."`
 - Write the method `drawPause()`
 - For now, leave it empty
 - In your `draw()` method
 - This method should be empty since you moved the code to `drawGameplay`
 - Replace this with if statements which evaluate `gamestate`. If it equals `"gameplay"` call `drawGameplay()`. If it equals `"pause"` call `drawPause()`.

Test Point ✓

Right now your game should pause... but never unpause. Does it work? At this point, add text output in the corner of the screen to display the current game state. This will help a lot in debugging later on!

Part B - Unpausing

- In `GameEngine...`
 - Modify the method `drawPause()`
 - If the user presses 'p' set `gamestate` equal to `"gameplay."`

Test Point ✓

Try this out. It isn't going to work very well... what do you need?

- Add in a timer to make sure the user can only press a key every so many seconds. You've done timers plenty of times before, so I'll leave that up to you!

Test Point ✓

At this point, your game should pause and unpause smoothly!

/* Extended Comment */

You might notice your text looks funny on pause, especially later on once you start drawing more. This is because we aren't erasing the background each frame, but it is drawing text over and over each frame. You may want to add in two parts of drawGameplay to drawPause. These are setting the background to black, and drawing all the objects in the game. We still won't call act or update, so your objects won't change. This is optional for now, but most people will want to add it in later!

Part C - Title Screen

- In **GameEngine...**
 - Add the method drawTitle()
 - Display an image for your title screen.
 - Your screen resolution in this room is 1440x900, so that makes a good image size if you're creating something in photoshop

Code to display a full-screen image

```
image(imageName, 0, 0, width, height);
```

- If the user presses a button (your choice), set gamestate to "gameplay"
- Be sure to display some sort of message so the user knows what to press to start the game.

Upgrades

Adding a “shop” system to your game

General Plan

This section outlines how to use a shop system to satisfy the upgrade requirement. There are many other solutions, but this is probably the simplest. Most of what you do with upgrades will be things you figure out on your own, since each game will have unique power-ups.

Our strategy will consist of three parts. First, we reward the player with credits for each enemy he kills. Second, we will implement a simple upgrade that you can only turn on or off by hardcoding it. Third, we will put the two together and allow the user to purchase the upgrade.

A Detailed Implementation

Part A - Money System

- In Player...
 - Add a class-level variable named **credits**
 - Write an accessor method called **getCredits()** that returns credits
 - Write a mutator method called **addCredits(int amount)** that adds amount to credits.
- In GameObject...
 - Add a class-level variable named **value**
 - Modify the **die()** method to do the following:
 - If **isAlive** is true
 - Set **curHealth** to 0
 - Set **isAlive** to false
 - Call the player's **addCredits** method, passing it value as a parameter

/ Extended Comment */*

This modification to the **die()** method will prevent you from accidentally getting paid twice if two of your shots kill the same enemy at once.

- In each enemy type
 - In the constructor...
 - Set value equal to **ENEMYNAME_VALUE** (the point value of that type).
- In GameEngine
 - Add a text output to display the player's credits

Test Point ✓

Your project should run, and each raider you hit will give you the number of credits you set it to be worth. You'll need to add a value to each enemy.

Part B - A Simple Upgrade

- In Player...
 - Add a class-level boolean variable named **tripleshot** to false
 - Modify your act() method, changing the code where you shoot. Instead of always creating two bullets, change its behavior based on tripleshot. You can even vary the cooldown time by setting it differently inside each condition. For example:

Code to display a full-screen image

```
if (tripleshot)    // The player has the upgrade
{
    objects.add(new PlayerShotBasic(x, y));
    objects.add(new PlayerShotBasic(x+12, y));
    objects.add(new PlayerShotBasic(x+24, y));
    shotTimer = PLAYER_MULTISHOT_COOLDOWN;
}
else              // Default shot
{
    objects.add(new PlayerShotBasic(x+4, y));
    objects.add(new PlayerShotBasic(x+20, y));
    shotTimer = PLAYER_SHOT_COOLDOWN;
}
```

- As a side note, your final project needs to include something that is not just a copy-paste of this upgrade. You can use a similar effect, but it needs to be different in some way.

Test Point ✓

Run your project. It should work EXACTLY as it did before. Now change your code to start tripleshot out as true. You should fire three vertical shots. Once you're done, set tripleshot's default value back to false in your code.

Part C - Setting the Upgrade

- In GameEngine
 - In the **drawPause()** method, add a line that calls the player's **buyUpgrades()** method.
- In Player...
 - Add a method called **buyUpgrades()**
 - If the user presses "1"...
 - Set tripleshot to true
 - Don't forget to use a keyTimer!
 - As a side note, your final project needs to include something that is not just a copy-paste of this upgrade. You can use a similar effect, but it needs to be different in some way.

Test Point ✓

Run your project. Fire - normal shots? Pause it. Press 1. Triple shot? Woo.

Part D - Buying the Upgrade

- In Player...
 - Add a method called **buyUpgrades()**
 - Modify your conditional to add two more requirements. It should now have four requirements joined by the && operator.
 - User presses the "1" key (done)
 - KeyTimer == 0 (done)
 - You should only be able to purchase the upgrade if **credits > TRIPLE_SHOT_COST**.
 - You should only be able to purchase the upgrade if **tripleShot** is false
 - Inside the conditional...
 - Subtract **TRIPLE_SHOT_COST** from **credits**.

Test Point ✓

Does it reduce the cost by the proper amount? Can you buy it when you have no money? Can you buy it multiple times?

Part E - Interface Update

- In GameEngine
 - Modify the **drawPause()** method
 - You should output some information about each upgrade on the pause screen. You will want to include...
 - Hotkey - What do I press to buy it?
 - Name - What is it called?
 - Price - How much money do I need?
 - Description (Optional) - What does it do?

Test Point ✓

Run it. Pretend you're a user, not the programmer. Is it clear? Are there any ways you could make this more intuitive? If it all works the way you intended, go on and add in new upgrades!