Space Banditos

Team: Aurora
Created by: Sean King - Matt Yefima - James Griffin - Andrew Kim

Overview of Game

- Game Play:
    - Key Features:
        - Turn based tactical RPG
        - 2D Characters / 3D World
        - Single Player
    - Win Conditions:
        - Destroying all enemy characters on the map in the normal missions
        - Defeating boss and his minion characters on final missions
- Genre:
    - Turn-based RPG
- Art & Music Design:
    - Art: Tiles and Stages are modeled in 3D with Maya and Blender. Characters are represented as 2D sprite models created in Photoshop and rigged in Unity.
    - Music for the entire game was created by Matt Yefima and his trusty guitar.
- Technical Platform:
    - Programming Language:
        - Space Banditos is programmed in C#
    - Engines:
        - Space Banditos is built using the Unity 5 Game Engine
    - Target OS:
        - Windows

Game Specification

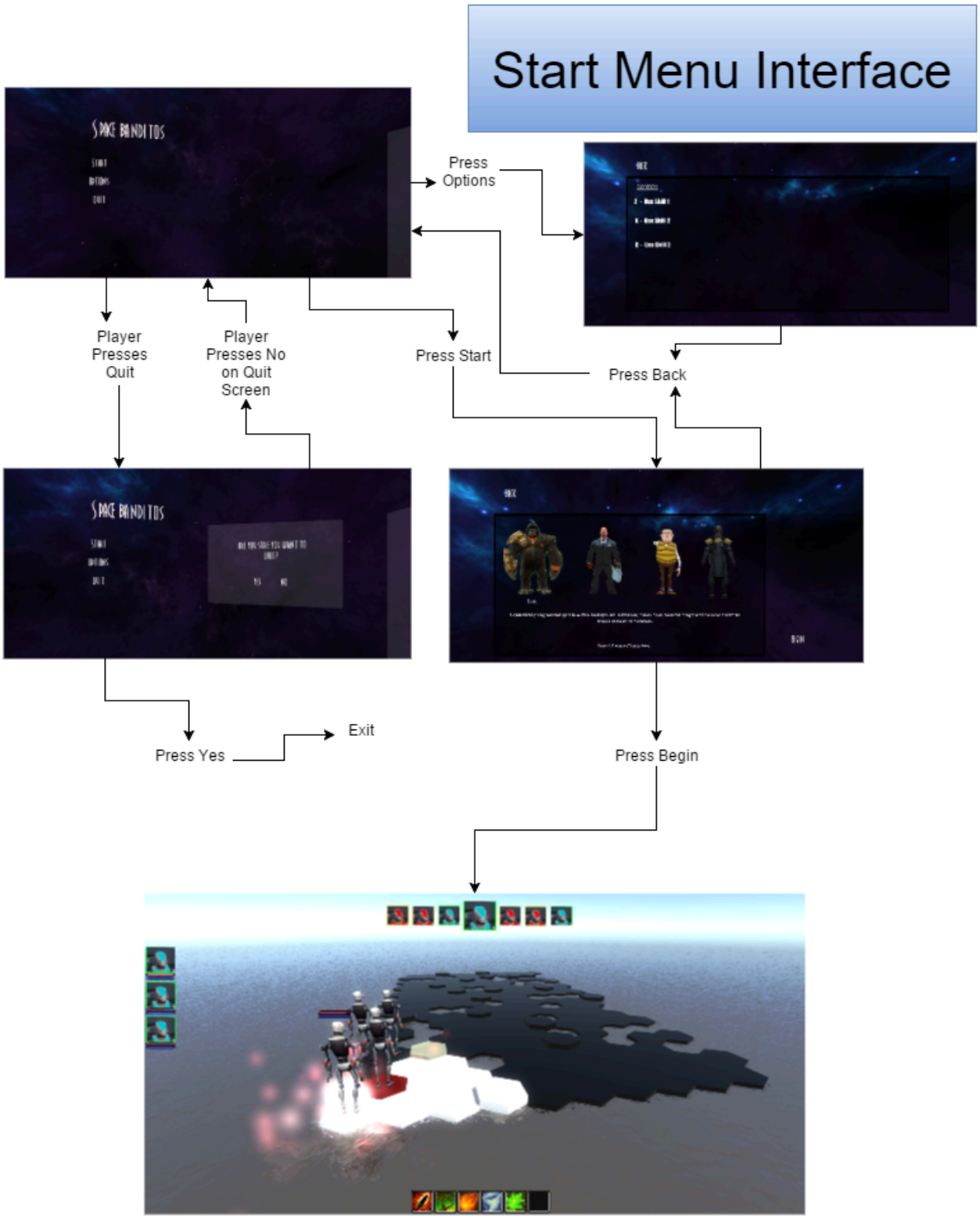
Rules and Mechanics:

- Beginning a Game
    - At the start of the game, the player is given a choice to select their team members along with their abilities. There are 4 character classes available, each with unique skill sets and utilities, from which the player may assemble a team of 3 characters. Upon selecting a character, the user is presented with set of talents that they may choose from to specialize that character as they see fit. Character classes and their talents are described later in the document.
- Selecting Character Talents
    - Talents are used to modify existing abilities, add new abilities, or enhance certain aspects of a character's performance. Choosing talents wisely allows the player to create teams with a better overall skill set to handle different situations in the game. However, the amount of points a player can spend in a tree is limited and talents cannot be adjusted once the game has started, so talents must be selected wisely. More information on each class' abilities and talents is available in each character's description.
    - Talents are selected by left clicking the icon to spend points, and right clicking the icon to remove points.
    - Each character class has 2 talent trees to specialize in. Players can spend all their points in either tree, or divide the points among both trees however they like.
    - At each tier in the tree, the player may choose to pick one of the two talents available from either specialization.
    - Although players can fully customize each character's talent trees, there are prebuilt talent sets to aid in quickly forming a team.
- Playing the Game
    - At the start of each stage, the player's party is placed on the opposite end of the map from the enemy characters.
    - Each turn, a character may move to a different tile, then use an ability. Abilities have cooldowns and resources costs associated with them.
        - Resource cost refers to the cost an ability has in order to be used. Each character has a fixed upper limit on resource spending per turn and each ability has a cost that consumes a portion of those resources. Resources are replenished every turn.
        - Cooldown refers to a time out period before an ability can be reused. This is to prevent powerful abilities from being used

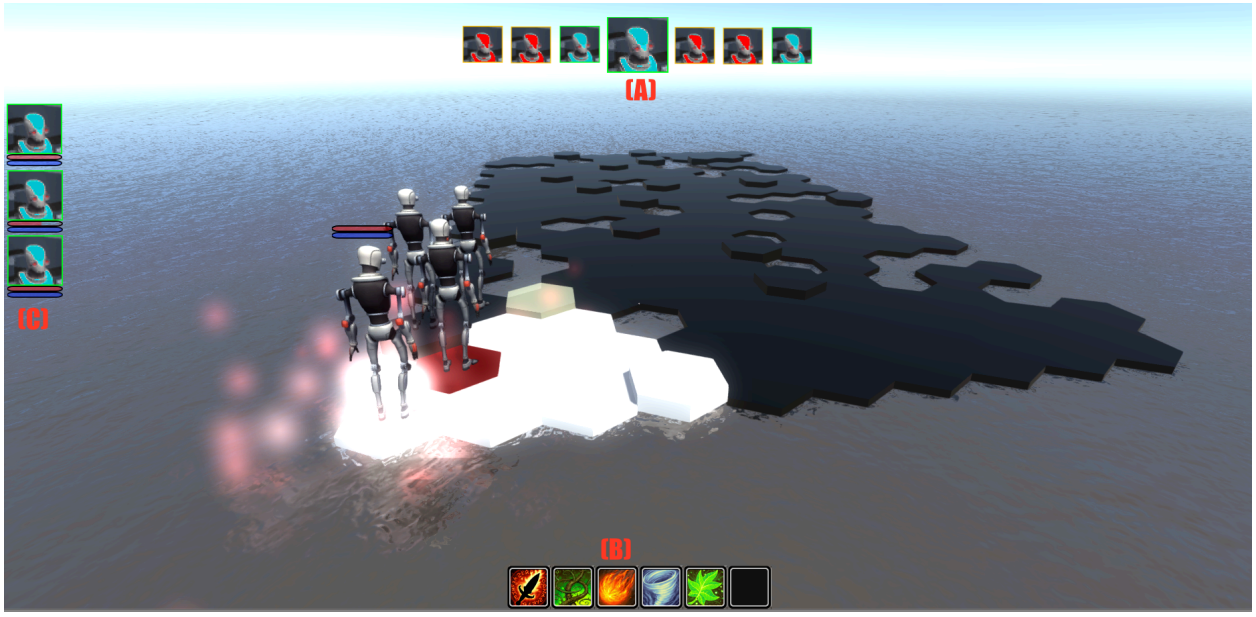
multiple times within a short time span while never using the rest of the character's abilities.

- Each level is completed by successfully defeating all enemy characters

Artwork and User Interface:



- User Interface Components



- In Game User Interface Components- These are components that the user will interact with once they start the game.
  - Turn Order Bar (A) - Located at the top of the screen, this element provides a sequence of character portraits showing the order of turns for all characters on the field. If the player clicks on a portrait, it will select the character.
  - Action Bar (B) - This element at the bottom of the screen shows all of the available actions for the currently selected friendly character and also contains buttons to open other parts of the user interface.
    - Abilities - By highlighting an ability icon with the mouse, a tooltip is revealed showing a description of the ability, its cooldown time, remaining cooldown if recently used, and its resource cost.
    - Menu - Brings up the main game menu, pausing the game and allowing the user to save/load a game state, adjust the game options, and quit to the main menu or desktop.
  - Characters Bar (C) - This element on the left side of the screen shows all of the current characters and clicking over a portrait will shift the focus of the camera onto a playable character based on which one is clicked. Some essential data is shown as well by this bar such as:
    - Health- This is the most important resource available to a unit; once this reaches 0 the unit becomes incapacitated and cannot perform any actions for the rest of the round.

            - Mana - Mana is the energy source that lets each unit use abilities; it recovers slowly at a static rate
    - Main Menu Screen - Upon entering the game, there will be a ‘Start Game’ button that will be centered in the middle of the screen with randomly generated maps that will showcase some of the environment conditions that the player will have to go through in game.
    - Selection Screen - This is the section that will guide the user through selecting their 3 characters out of the 4 available to play in the game.
        - The UI layout will be 4 vertical columns equidistant from each other and within it will have a display of each character at the top to give a visual representation of the current character they are on.
        - Just to the right of the character’s picture, will contain a checkbox to tell the player whether or not they have that current character they’re on selected or not.
        - Below the character’s picture and the checkbox will contain a talent tree with 5-7 various talent tiles that the player will be able to select from. When the mouse hovers over each talent tile, a popup box will appear that will briefly explain that specific talent.
        - When traversing from one character column to another, the columns that aren’t in focus will have a grey overlay so the current column will possess all the focus.

The Story So Far:
In a galaxy that has been stricken with an overabundance of outsourced jobs and has become rampant with competing gangs, four broke, semi-talented heroes roam around together in search for a stable living to supplement their increasingly expensive and frivolous spending habits by bringing in those gangs for a bounty - dead or alive.

The Tank’s Backstory:

http://mondaynightcombat.wikia.com/wiki/File:Cheston_Concept_Art.png
A genetically engineered gorilla with a background in finance; times have become tough and he now crunches bones instead of numbers.

The Medic's Backstory:

http://danmalone.deviantart.com/art/BORSI-model-sheet-31329793
A boring, by the books doctor fella that has been searching for the keys to those handcuffs for far too long.

The Tech's Backstory:

http://www.characterdesignpage.com/blog/valerio-fabbretti
This is Jeff. You wouldn't invite him to a party unless it's a gang raiding party. His engineering ability is unparalleled because of an abusive father that kept him in the basement until his 40's with nothing but an Arduino to pass the time.

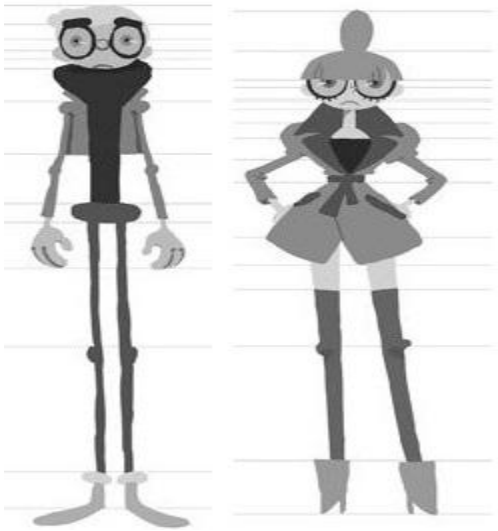
The Assassin's Backstory:

http://www.starwars.com/tv-shows/clone-wars/to-catch-a-jedi-concept-art-gallery
Jamal left behind a very lucrative job as a professional cosplayer to become a bounty hunter because the thrills of historic cosplay competitions became mundane, so he decided to join the ranks of the Space Banditos.

Head Honcho Number 1 (Dubbed: The Shorties):

http://robertvalley.tumblr.com/image/26222064258
From a world full of tall legged, monotone colored people, this man was picked on relentlessly because of his embarrassingly short legs. He decided to get back at his world and all other worlds for picking on him with the help of his short-legged crew.

The Shawties Minions:



http://theconceptartblog.com/2013/09/12/annie-curta-de-graduacao-da-gobelins/
Not important.

Head Honcho 2 (The Green Machines):

http://www.bradfitzpatrick.com/wp-content/uploads/cricket-character-sheet.jpg
With great wrist cuffs comes great responsibility, but Green Bug Guy took it as an omen to gather a gang of fellow well fashioned wrist cuff enthusiasts to take over their desert world.

The Green Machine's Minions:

http://www.davpunk.com/portfolio-item/quick-model-sheet

Head Honcho 3:

http://theconceptartblog.com/2013/01/29/mais-artes-produzidas-para-o-filme-paranorman/

Mug shot of a Grandmother that sends her own grandchildren out to do her nefarious bidding.

Minions:

http://theconceptartblog.com/2013/01/29/mais-artes-produzidas-para-o-filme-paranorman/

The unfortunate and apathetic grandchild that started his life of crime early on at his grandmother's crime syndicate daycare.

Characters:

The game features 4 different character classes from which the player can select 3 at a time to form a team. Each character has a core ability set that is fixed around that character's primary role. Talents can then be selected to either improve this role, or to give the character a more diversified skill set, allowing for variation in not only team setups, but variation in ability options for each combination of characters.

- The Tank
    - Design philosophy
        - The Tank's main role is to serve as the party's defensive pillar by taking in as much of the incoming damage as possible. Having a tank allows other characters to focus on maximizing their own damage output while being assured that the damage they receive from enemy characters is minimized.
        - The Tank's secondary role is to become an offensive damage dealer specializing in close range damage dealing and debuffing.
    - Tank Core Abilities
        - Taunt - Forces all targets in a given range to focus their attacks on the tank for a set amount of turns. This ability has a short cooldown and low resource cost.

            - Taunt is the main ability that will be used to prevent damage being dealt to other party members by forcing enemies to focus on the Tank.
        - Persistence - For the next 6 turns, any damage dealt tank has a 60% chance to be avoided completely. Persistence has a low resource cost but a long cooldown.
            - Persistence gives the tank additional survivability through a possibility of avoiding incoming damage. It is best used when the tank is already at low health, providing an opportunity for recovery.
        - Slam - Deals damage in a radius around the character and stuns them for a set amount of turns. This ability has a long cooldown and a high resource cost.
            - The best uses for this skill are to actively prevent upcoming enemy turns or allow a recovery period when the party is in dire straits.
        - Weaken Defense - This ability deals damage and lowers the armor of the target, making it more susceptible to damage from other party members. It has no cooldown and a low resource cost.
            - Ideally, this ability is kept active as often as possible to maximize the party's damage output.
    - Tank Talents
        - Enrage - When health drops below 50%, damage is increased by 20%. This is a passive ability and has no cost or cooldown.
            - This ability is meant to encourage keeping the Tank as the defensive character for the team, but presents a risk/reward trade off for keeping the tank at low health.
        - Surrounded - If more than one enemy is within 2 hexes of the Tank, deal equal damage to all targets within 2 hexes with every attack. This is a passive ability and has no cost or cooldown.
            - Similar to enrage, Surrounded encourages keeping the Tank at the forefront of combat but provides a large damage reward for doing so.
        - Slam Upgrade - Halves the stun duration but doubles the damage Slam deals. This ability replaces the default Area of Effect skill for the tank. The cost and cooldown are not changed.
            - This talent makes the Tank more of a damage oriented character than a controlling, defensive one. Taking this

upgrade is particularly useful when other party members provide enough crowd control to handle multiple enemies.

- Backup Shield - When health drops below 30%, gain a shield equal to 50% of the Tank's maximum health. This passive ability has no cost, but can only occur once every 3 turns.
    - This ability makes the Tank able to withstand more damage and is useful if the player wants to feel safer in their play style or perhaps use the medic more aggressively, letting the Tank last longer without support.
- Front lines - For every enemy within 3 hexes of the Tank, restore 3% of the Tank's health per turn. This is a passive ability and has no cost or cooldown.
    - Similar to Backup Shield, this talent makes the Tank more self sufficient when surrounded by enemies.
- Rally - For every friendly member within 4 hexes of the Tank, reduce damage taken and increase damage dealt by 5% for all nearby party members. This is a passive ability and has no cost or cooldown.
    - This talent encourages the player to use the rest of the team more aggressively when a defensive Tank is present in the party and helps to balance out the loss of damage the Tank causes to the team by boosting the entire party's damage.

- The Medic
    - Design Philosophy
        - The Medic's main role is to heal the other party members and provide support through buffing. This keeps the party alive and improves both their damage output and defenses.
        - The Medic's secondary role provides defensive debuffing and crowd control abilities, weakening the enemy characters and reducing their strategic options. It also decreases the need for direct health recovery by preventing incoming damage, allowing the Medic to be more flexible in its ability choices.
    - Medic Core Abilities
        - Heal - Restores a great amount of health to a single party member. This ability has no cooldown and a low resource cost.
            - This ability best used when a single party member is sustaining most of the damage.

        - Area of Effect Heal - Restores a moderate amount of health to all party members within the target area. This ability has a high cost and a short cooldown.
            - This ability is similar to the regular Heal spell, but is used to recover the health of multiple party members. This is useful for teams consisting of multiple melee characters.
        - Weaken Offense - Reduces the damage output of a single target. It has no cooldown and a low resource cost.
            - This spell is complementary to the tank's Weaken Defense ability, but in the theme of a medic helps the party sustain less damage.
        - Incapacitate - Causes an enemy to be unable to act for a set amount of turns. Only one enemy can be incapacitated at a time. It has a low cooldown and a moderate resource cost.
            - This ability is used to remove an enemy from combat temporarily, making larger or more problematic groups easier to deal with.
    - Medic Talents
        - Anatomy - Buffs the friendly target to increase their damage and healing for 4 turns. This ability has a low cost and no cooldown.
            - This ability allows the Medic to boost the capabilities of any other party member, but is not particularly useful on a defensive Tank since it does not boost defensive stats.
        - Med Station - Heals all friendly characters within 5 hexes of the Med Station for up to 10 turns. This ability has a high cost and high cooldown.
            - This talent promotes keeping the party close together and allows the Medic to use more aggressive buffing and debuffing skills instead of only using healing abilities.
        - Balance (Heal upgrade) - Heals a friendly character and increases their armor for 3 turns. Deals damage to the enemy target closest to the healed party member and lowers their armor for 3 turns. This ability replaces the standard Heal and does not affect the cost or cooldown.
            - Balance lets the Medic contribute more effectively to the team by improving the armor of those healed and providing some light damage support for the team.

        - Blood Donor - Deal damage to the enemy target, restoring a portion back to the medic as health. This ability has a medium cost and no cooldown.
            - This talent lets the Medic function as a damage dealing character while improving its survivability with the health restoration effect.
        - Lethal Injection - Ranged ability that deals damage and reduces both the damage and armor of the target. This ability has a low cost and no cooldown.
            - This is the primary damage dealing ability for the offensive Medic due to its low cost and debuffing effects, preserving resources for other abilities while providing support to the rest of the party.
        - Area of Effect Heal Upgrade - Replaces the Area of Effect Heal ability's healing with damage and reduces the armor of all enemies hit. This ability has a high cost and low cooldown.
            - This talent is useful for play styles involving multiple medics or for those involving a more aggressive overall approach.

- The Tech
    - Design Philosophy
        - The Tech's main role is to provide utility and damage output through the use of pets, gadgets, and so on.
        - The Tech's secondary role is to improve it's own damage output and crowd control foregoing the use of pets.
    - Tech Core Abilities
        - Sentry - Places a stationary gadget that periodically deals damage to enemies in range. This ability has a low resource cost and no cooldown, but only a limited number can exist on the field at once.
            - This ability is used not only for damage output, but can be used to block off paths if they are wisely positioned.
        - Trap - Locks the target to its current position on the field for a set amount of turns. The enemy is still able to act but is unable to move. This ability has a low cost and a short cooldown.
            - This ability is used as crowd control by keeping targets out of the range from which they can deal damage to the other party members.
        - Snipe - Deals a heavy amount of damage to a single target. This ability has no cooldown but a high resource cost.

            - This is the Tech's primary damage dealing ability.
        - Repair - Restores a low amount of health to the target and increases its damage output for a set amount of turns. Health restored to pets and gadgets is greater than that restored to party members. This ability has no cooldown and a medium resource cost.
            - This ability is used not only to maintain gadgets, but to help improve party members' damage output.
        - Detonate - Destroys a gadget on the field, dealing damage to enemies around the gadget. This ability has no cooldown and a low cost.
            - This is best used when a gadget has sustained more damage than is worth repairing, allowing the Tech to squeeze a good amount of utility out of the gadget before it gets destroyed by the enemy forces.
    - Tech Talents
        - Healing Sentry - An additional sentry that can heal friendly targets. This ability has a low resource cost and no cooldown, but only a limited number can exist on the field at once.
            - Healing Sentry is effective for letting the Medic focus on providing buff support to the party instead of casting heal spells.
        - Barrier - A new gadget that provides a shielded area in which party members receive reduced damage. Only one barrier can be active at a time. This ability has a low cost and no cooldown.
            - In conjunction with the Healing Sentry, this ability helps make the Tech a partial replacement for the Medic.
        - Robot pet - A new ability which summons a pet that can be used as the party's tank. This ability has a low cost and no cooldown.
            - This is meant to be kept active as often as possible as a full replacement for a party without a defensive Tank.
        - Sticky Grenade - A new ability which attaches a timed grenade to the enemy target. After a set amount of turns, the grenade will deal a heavy amount of damage in an area around the target plus extra damage based off the damage the target received while the grenade was active. This ability has a long cooldown but a low resource cost.
            - This ability is effective both in dealing with a large single enemy or with groups of enemies. It is ideally cast on a boss

or other high health character who is being focused down by the party.
        - Improved Snipe - Upgrades the Snipe ability by reducing its resource cost.
            - This allows Snipe to be used consistently over a longer period and in conjunction with other high cost abilities, greatly enhancing the tech's damage output.
        - Shock Trap - Places a trap on a hex that deals damage and stuns an enemy that comes within 1 hex of the trap. This ability has a low cost and no cooldown, but only 2 traps may be active at a time.
            - This ability lets the Tech provide crowd control support to the team while dealing damage, but the traps must be strategically placed to force the enemy party to trip them.
- The Assassin
    - Design Philosophy
        - The Assassin's main role is to provide high damage output at melee range. This is a risk-reward play style, as the assassin is a fragile character and cannot sustain a great amount of damage, but the Assassin's melee damage is unmatched by other party members.
        - The Assassin's secondary role is as a ranged utility class. This specialization keeps the assassin out of harm's way, but the Assassin becomes more focused on support and crowd control as opposed to outright damage dealing.
    - Assassin Core Abilities (Assuming more powers)
        - Epidemic - Mark a target so that it explodes after 2 more of the Assassin's turns, dealing damage in an area around the target. All targets hit by the original explosion of epidemic also become marked, but their explosions do not mark more targets. This ability has a high resource cost and no cooldown.
            - Epidemic is used not only as a primary Area of Effect attack for the Assassin, but it also provides a debuff to be used by the Early Demise talent.
        - Cripple - Deal low damage to a target from a distance and reduce its movement range to 1 hex for 3 turns. This ability has a low resource cost and no cooldown.
            - This skill helps to keep enemies away from fragile team members, but is not useful as a primary damage skill.
        - Fade - For the next 3 turns, any damage dealt to the Assassin is avoided. This ability has a low resource cost and a long cooldown.

            - Since the Assassin is a low health character and thus likely to be in dangerous situations often, Fade provides a way to stay in range for dealing damage while providing time to recover.
        - Shiv - Stab the target, dealing low damage. This ability has a low resource cost and no cooldown.
            - Shiv is the Assassin's primary damage dealing ability and is mainly used when there are not enough resources for other skills to be used. It does not deal a large amount of damage on its own, but becomes very powerful when combined with the Assassin's damage boosting talents.
    - Assassin Talents
        - Shadow Drift - Relocate to a hex adjacent to any other character on the map. Teleporting to a friendly character restores 15% of the Assassin's health. This ability has a high resource cost but no cooldown.
            - Shadow Drift provides an escape and return ability for the Assassin. If the player predicts the Assassin may be in danger in the next few turns, they can retreat safely and return to the fray once they are healed.
        - Early Demise - If a target is marked with epidemic, deal damage to that target equivalent to all the Area of Effect Damage it would deal upon explosion. Removes the Epidemic mark from the target without exploding. This ability has a low resource cost and no cooldown.
            - Early Demise provides an attack rotation for the Assassin. It is best used on targets who are afflicted by the second wave of Epidemic after the original mark explodes.
        - Singled Out - If an enemy target has no allies within 3 hexes, all damage dealt by the Assassin to that target is doubled. This is a passive ability and has no resource cost or cooldown.
            - This talent boosts the Assassin's damage output against any targets that have been separated from their allies. It can be used in strategies that leave the boss as the final character, or in conjunction with crowd control skills that leave enemies isolated.
        - Lethal Blow - Deal damage to a target from a distance and expose its vulnerabilities, increasing the chance of critical strikes against

this target by 15% for all party members. Lethal Blow has a low resource cost and no cooldown.
    - This ability deals low damage, but debuffs the target to provide an overall damage increase for the entire party.
- Pierce (Shiv Upgrade) - Pierce replaces Shiv with a ranged attack that locks the target in place for 3 turns. The cost and cooldown remain unchanged, but the damage dealt is increased.
    - Pierce transforms the Assassin into a ranged character. The damage is increased to compensate for the inability to apply melee range effects such as Epidemic.
- Efficiency - Ranged ability that marks a target for 5 turns. If the target is friendly it restores health to the target, and deals damage if the target is an enemy. Any ability cast on this target will restore 5% of the caster's resources. This ability has a low cost and no cooldown.
    - Efficiency provides utility to the entire team by effectively reducing the resource cost of all abilities that are cast. It is helpful to healers as well as damage dealers since it can be used on friendly targets.

CutScenes:

Before each level loads, a page will show up before you start playing the game to create a backstory for each gang and the bounty you'll receive for taking them in (dead or not alive). This backstory style will be similar to that from Cowboy Bebop.

Levels:

There will be 3 levels in this game, each with a specific gang that have abilities that complement their environment. All levels will be randomly generated in order to increase the replayability of each play through so it will be highly unlikely that the player will be able to use the same strategy each time he/she plays the level. The goal of each level is to kill all of the enemies (boss and all their minions).

- Level 1: Cantina
- Level 2: Junkyard
- Level 3: Spaceship

Scripts: Does not apply

Technical Specs

Language:

- Language
    - The game's code will be written exclusively in C#
- Code Environments
    - All game code will be written, edited, and compiled in the following IDEs
        - Visual Studio 2015 Enterprise Edition
        - Monodevelop
    - The choice between these two IDEs is to each programmer's discretion.
- While there is no emulator to test and run this game, Unity does come with a tabbed view where the programmer can enter the game scene and essentially 'play the game'.

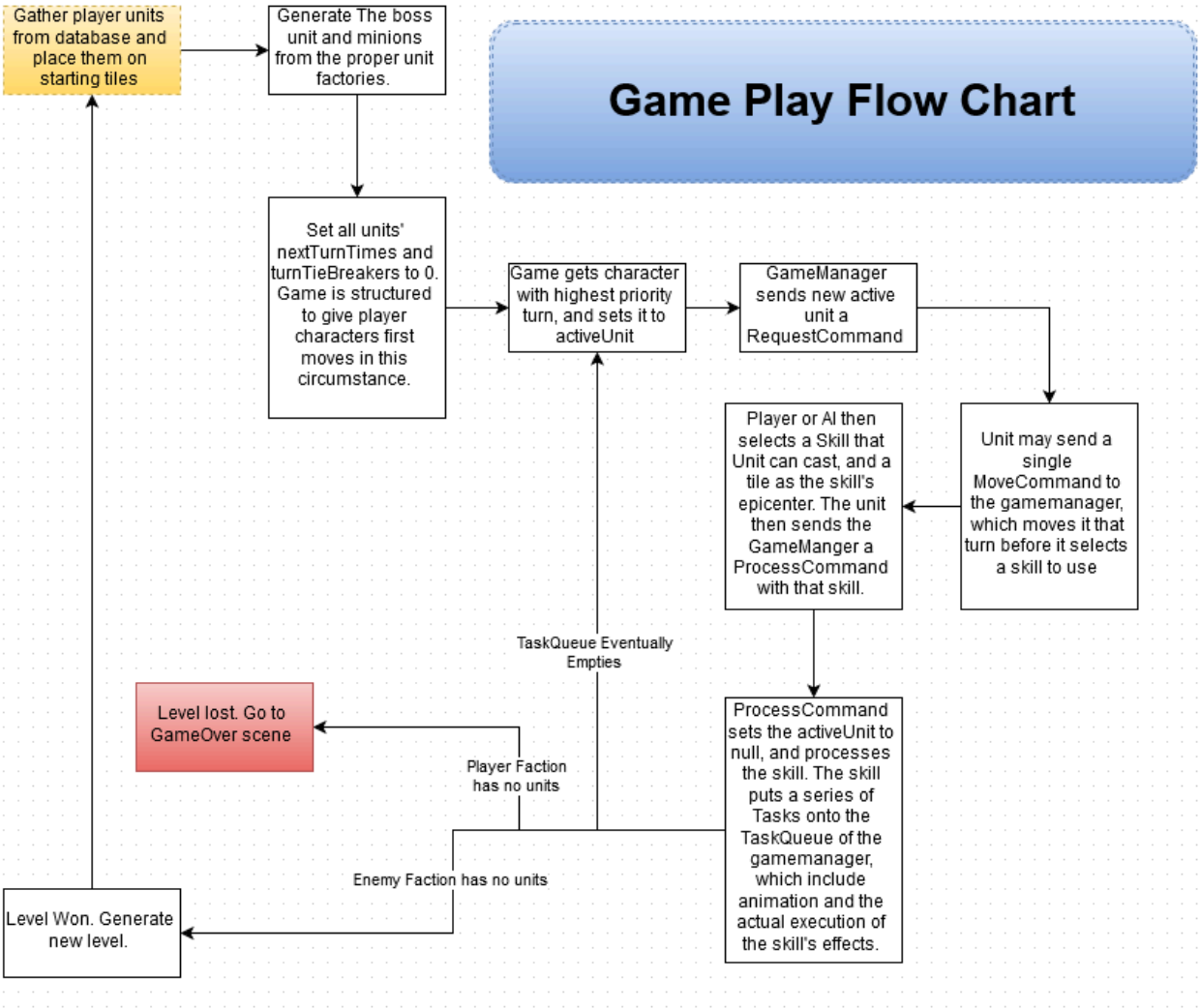
Engine:

We will be creating our game in **Unity 5**.

- Overview of Unity 5
    - Unity is a popular and well supported cross-platform game engine and environment.
    - The Unity Engine organizes games into different scenes
        - Each scene contains an array of distinct entities called GameObjects
    - Unity is a component oriented engine, and GameObjects are little more than empty shells that acquire unique functionality from an internal array of Component classes.
        - Each Component Class represents a small piece of functionality, such as audio output, physics, rendering, and custom scripting.
        - Component scripts are written in C#, and are refered to as MonoBehaviors.
    - Interface
        - Unity, like most other IDE's, has a view of the project hierarchy, views that control properties of components, and asset control. But what makes Unity so powerful and the reason why we chose it as our game engine is that because its interface allows for such rapid development. Its Game View tab makes it easy to add point of view camera positions to quickly see what the player will see when the developer enters 'Play Mode'.

- Game view control bar allows for things such as windowed mode, fullscreen mode that creates a more immersive effect of the game being developed, and the statistics of the game's performance on the current machine it is being run on.
- Scene View is basically a sandbox into the world in which we created. We are able to view the scene (or 2D map in our case) in its entirety. This is where we can place game objects in specific locations and camera views for the player. One nuisance that is noteworthy is that in order to navigate through the scene view's environment, a mouse is essential. Navigating with a trackpad on a laptop is horrible.
- Inspector View is the place where all the properties, scripts, sounds, graphical elements, and meshes are located for the currently selected GameObject. It is where we can add new AI features to specific objects, change the object's looks and colors and physically change the game camera's XYZ positioning.

Control Flow of Gameplay:



Data Structures and Classes:

- **GameManager Class**
  - **Overview**:
    - GameManager is a singleton class designed to easily collect and maintain information about the current scene, and provide easy access to this information to other game agents.
    - GameManager maintains an accessible grid representation of the current scene's map of hexagon tiles.
    - GameManager uses its grid representation to generate navigation paths and movement ranges for Agents who query for them.
  - **Tile Grid System**:
    - GameManager maintains a 2-D list of references to Tile components.

- Each Tile is a component class attached to each of the actual tile GameObjects in the scene. They will be explained in more detail later, but each contains an int vector caching their location in the GameManager's grid.
- The first list of the grid represents columns, and each element is a list representing a row of tiles.
- tiles[x][y] returns the hexagon tile x spaces to the right and y spaces forward in the tile system. If there is no tile at that particular space, tiles[x][y] is null.
- List<Tile> Neighbors(tile):
    - A function which returns a list of up to 6 hexagon tiles directly next to the tile used as an argument.
    - Due to the nature of hexagons being used in an xy coordinate system, all odd rows of tiles are offset to the right so that they can interlock.
        - All tiles have neighbors immediately to the left and right of them regardless of their offsetness.
        - Non-offset tiles have neighbors above and below with their x indices displaced by -1(left) and 0(right).
        - Offset tiles have neighbors above and below with their x indices displaced by 0(left) and 1(right).
    - If a there is no tile at a calculated neighbor space as a result of its index being out of bounds or the element being null, it simply wont be added to the list.
    - Neigbors(tile) is required for all traversals of the grid system, due to the complications of hexigons that it abstracts from.
- List<Tile> FindPath(Tile,Tile)
    - A function which generates a smallest-cost series of legal tiles to move through on a path from the first tile to the second.
        - Legal tiles are tiles that actually exist, are not obstacles, and do not already have a character occupying them.
    - Every time a Character is commanded to move to a legal hexagon, the path list is used to animate its movement.
    - FindPath uses a standard A* search algorithm.
        - PathNode is a private inner class used for this algorithm, each instance representing a tile, a

previous Node in the path, its cost from the start, and its estimated cost to the end.
        - FindPath uses a sorted list for its open set, and a Dictionary that maps Tiles to PathNodes for its closed set.
            - The dictionary allows quick checking to see if a tile already exists as a PathNode and whether the current node is a better predecessor.
        - Each neighbor move has a cost of 1.
        - The heuristic is the euclidian distance between the physical position of the tiles in the game map.
            - Each hexagon model is exactly 1 game unit across so this heuristic works.
    - Set<Tile> TilesInMoveRange(Tile,int n)
        - A function which generates a set of tiles reachable from the current tile in n moves.
        - This is accomplished with a recursive depth limited search.
            - The DLS is given a maximum depth of range of n
            - Each recurrence of the DLS has access to the same Set, which it inserts the current tile of the search into.
            - When the DLS runs out of moves, the Set of reached tiles is finally returned to the original caller.
        - The Set represents a Character's current list of tiles it can legally move to this turn based on its move speed.
- **Character and Turn System**
    - GameManager keeps track of a list of the characters currently combatting on the map.
    - GameManager manages the order of turns. Each character has a nextMoveTime variable associated with them, and their position in the list is ordered by that, with ties being broken by who actually moves least recently.
    - GameManager can be queried by a character to obtain the nearest enemy or ally, or a list of enemies or allies within a given range(including the whole map).
    - GameManager is ultimately the class which tells an Agent(player or ai controlled) that it is its turn, and requests an action.
        - The Agent, after being given this request, and deciding its action, implements its action by calling GameManager's ProcessCommand(Command) method.

        - Void ProcessCommand(Command):
            - The method eventually called by a Character to dictate its chosen move and end its turn.
            - The command can be a move command, or spell command.
            - Command is a class which describes to GameManager the the Agent's desired action.
            - If the command is illegal, such as a move command to an out of bounds tile, the method fails and the Character is told to make another decision.
            - if the command is legal, it will be carried out by the appropriate system, animated, and following the animation the character's nextMoveTime is incremented appropriately. The next character is selected to take a turn.
            - In the event that the command's execution results in the defeat of the player's team(there are no player characters in the unit list), the scene will transfer to the gameover screen.
            - In the event that the command's execution results in the victory of the player's team(there are no enemies in the unit list), The scene will transition to the between stages scene.
        - Task Queue
            - Game skills and movement carry out their animation as a queue of tasks.
            - Each task represents either an animation or an implementation of a command.
            - While there are tasks in the queue, the GameManager will go through them, not allowing units to send ProcessCommands.
- **Tile Class**
    - **Overview**
        - Tile is a MonoBehavior component class which is attached to each individual hexagon tile GameObject, and is used by the GameManager to reason about the structure of the interlocking tile grid system which makes up a scene.
    - **Specifics**
        - Each Tile component contains a 2d Vector of int's to keep track of its position in the Game Grid.
            - If GameManager points to a particular tile as tiles[x][y], then this vector will be <x,y>.

            - When GameManager is given a query such as find path or get neighbors of a tile, this internal vector is used to calculate the starting location in GameManager's grid.
        - Each Tile has a Unit variable
            - Unit is the class representing a character.
            - If there is a character occupying this tile, this will be a pointer to it.
            - If there is not character occupying this tile, this will be null.
- **Unit Class**
    - **Overview**
        - Unit is a MonoBehavior component class attached to GameObjects representing the characters on the game map. It references and is referenced by the Tile that it is currently occupying, and is kept track of by the GameManager.
    - **Specifics**
        - Each Unit has a Tile variable to reference the tile it currently resides on.
            - This variable is never null. Either a Unit is dead or it's on a tile.
            - The tile a Unit is pointing always has a reference to that Unit.
            - Both references are broken when a Unit moves to a new tile after a GameManager ProcessCommand call.
        - Each Unit maintains a List of SkillContainers
            - Skill is a class which represents different spells and attacks, their cooldowns, costs, ranges, aoe's, etc.
            - SkillContainer is a wrapper class for Skills.
                - The DataManager singleton maintains a list of constant immutible Skills that are designed before hand.
                - SkillContainer keep track of additional mutable information regarding skills, such as the current cooldown and current augmentations.
            - When a player is selecting a character's moves, this List is used to generate Buttons in the game scene's GUI
                - Each button generates a Command instance for that skill which will be used to call the GameManager's ProcessCommand method.
                - If a skill requires additional information such as a target hexagon or target character, the control context

of the current scene will switch to a tile selector to aquire a target. After the target is selected, that information is also passed into the Command instance, and delivered to the GameManager.
        - Ai controlled characters will skip this step and directly put the target coordinates into the Command instance.
    - Each Unit uses a loose collection of private floats to represent its various stats.
        - These include maxHP, curHP, resources required for skills, etc.
        - Each stat has a corresponding public get wrapper, which returns not only the base stat, but gathers the bonus values attached to all current effects affecting the Unit at that moment and returns their sum.
    - Each Unit has an array of EffectContainers
        - Exactly like SkillContainers, but for current effects on the character
            - The Effect class is an immutable template like Skill and exists in a single array maintained by the DatabaseManager singleton.
            - Effects need a Wrapper to convey temporary variables, such as remaining duration.
        - Each effect has an OnTurnStart(Unit), OnTurnEnd, OnMove, and OnSkillUse method that can be empty. Each time one of these events occurs, the Unit cycles through the SkillContainer List calling the appropriate effects.
        - At the end of a Unit's turn, if the EffectContainer is maintaining a cooldown int, this cooldown will be decremented, and the EffectContainer will be removed in the event that the cooldown is less than or equal to 0.
    - Each Unit has an int to represent its faction.
        - 0 is players team
        - 1 is enemy team
        - The factions are tested for equality by the GameManager when isEnemy(Unit,Unit) or isAlly(Unit,Unit) is called. if they are not equal, the Unit's are enemies.
- **Skill Class**
    - **Overview**

        - Each instance of the Skill class defines a particular skill in the Database
        - Skills have no direct connection to the Unit calling them, and the user of the skill must be passed as an argument to a Skill's methods.
    - **Specifics**
        - All skills are part of the DatabaseManager's array of skills, and each Unit that knows a particular skill shares a reference to the same instance.
        - Unique data regarding each Unit's relation to that skill comes from the SkillContainer wrapper class which maintains information such as cooldown. Unit's cannot mutate a raw Skill instance in any way.
        - Skills have the following variables:
            - Cooldown : int
                - Cooldown is the amount of cooldown placed on the SkillContainer after a skill is used.
            - TargetType : enum
                - Tracks if the skill targets only self, only allies, only enemies, all units, or nothing.
            - Range : int
                - The default range of the spell.
                - self targeting spells ignore this
            - Cost : Map of variable names to floats
                - Represents the cost of using the spell
            - CustomRequirements : bool delegate
                - If this delegate is empty, the requirements checked before casting a spell is if the cost is payable. Otherwise it checks all functions belonging to this delegate.
            - CustomEffects : void delegate
                - All functions belonging to this delegate are performed when this skill is performs.
            - areaOfEffect : int
                - the number of tiles within range of the target tile which will also be hit by the spell. GameManager will perform its depth limited search algorithm to see which tiles are hit.
            - OnUse : void(Unit user, Unit target) delegate

            - When the skill is performed, all functions in the delegate are called for each target in the skills area.
        - Skills have the following methods:
            - GatherTargets(user,tile,SkillContainer wrapper) : List<Unit>
                - Gather targets that will be affected by the spell when targeting a particular tile. Takes into account the aoe of the skill, and uses the TargetType of the skill and the faction of the user to calculate which units are viable targets. The user's SkillContainer can be passed as well, in case it has an effect changing aoe.
            - Perform(user, tile, SkillContainer wrapper) : void
                - User's resources will go down according to the cost map of the skill. Uses GatherTargets on the tile. For each unit in the returned list of GatherTargets, the OnUse delegate is performed. The user's SkillContainer can be passed as well, since it may contain effects changing the values and effects of different skills.
                - The animation and execution of the skill is divided into Tasks, which get added to the GameManager's task queue in order of their desired execution.

- **Task class**
    - **Overview**
        - Tasks represent atomic packets of code responsible for being built up to form the implementation of animations and skills.
        - Each skill generates a list of tasks responsible for showing the player animations, as well as executing the effects of the skill
    - **Specifics**
        - Task is a base class with extended classes to represent specific tasks, such as movement, animation, and skill execution.
        - When a task is popped off the queue, it's OnEnter function is called, which sets up important variables.
        - While the task is being performed, each frame, the task's bool : OnUpdate() method is called. This returns false if the task is still running, and true if it has finished.
        - When OnUpdate() returns true, the task's OnExit() is called, to clean up loose ends related to the task. If there is another task in the queue, it will then be popped of and the process will repeat.

- **DatabaseManager class**
  - **Overview**
    - DatabaseManager is a singleton which keeps track of all global data relevant to more than a single scene.
    - Different from GameManager, which maintains global data which only lasts a single scene, and is refreshed each new scene.
    - DatabaseManager maintains premade information on skills, class types, and talents, as well as the current makeup of the team during a game's run.
  - **Skill Database**
    - DatabaseManager maintains a list of pre-scripted Skills. This list is created at runtime, and remains immutable for the duration of the program. Any changes to skills must come from SkillContainer wrappers pointing to an entry in the database.
    - These skills are scripted as ScriptableObjects and kept in the "Assets/Resources/Skills" directory. All entries in this folder make up the total array of skills in the game.
    - Element 0 in the array is the generic attack skill.
    - Element 1 in the array is the generic move skill.
    - Elements greater than 1 are all normal skills.
  - **UnitClass Database**
    - DatabaseManager maintains a list of UnitClasses. These are instances of the UnitClass class which define all aspects of the multiple classes of the game. These include talents and base stats.
    - When a new game is started, the list of classes to pick from for a team comes from this list.
    - After the team composition is selected, each Unit of the team is generated from the corresponding UnitClass's factory method.
  - **Effect Database**
    - DatabaseManager maintains a list of serialized Effect instances.
    - Like Skills, Effects are immutable and once created at the start of runtime cannot be directly edited, only referred to by a mutable EffectContainer.
    - These effects are ScriptableObjects and stored in the "Assets/Resources/Effects" directory.
  - **Dynamic Player Party UnitDatabase**
    - The DatabaseManager also maintains a list of dynamic elements that persists through the game : the Units composing the players party.

        - At the beginning of a battle scene, the GameManager puts pointer to the elements of this list in its Unit list. Any changes to the player Units in that list are thus reflected in this party list.
- **UnitClass class**
    - **Overview**
        - UnitClass is a factory class which produces Units.
        - DatabaseManager maintains a list of UnitClasses to represent the available Characters that the player may select from when building their party from the beginning screen.
    - **Specifics**
        - UnitClass contains a premade serialized Unit as a template. When its factory creation method is called, a new instance is cloned from that Unit.
        - UnitClass also contains an array of Talent Choices.
            - At the start of the Game, when you select a character, you are given a list of Talents to choose from.
            - Each talent chosen represents a permanent effect on the unit, a new talent the unit will be able to use, or adds a tag to the unit, which skills will query to see if they've been upgraded for that character
        - UnitClass instances are serialized and stored as assets in the "Assets/Resources/UnitClass" subfolder.
    - **TalentOption Inner Class**
        - Each TalentOption contains a void(Unit) delegate.
        - Has a bool toggle, representing if the talent has been chosen or not.
    - **Factory Method**
        - Once the maximum number of choosable talents have been toggled through the opening menu, the Factory Method can be called.
        - Instantiates a clone of the serialized unit. Then goes through the array of TalentOptions, and performs the delgates of each toggled talent. The functions in these delegates are scripted to permanently alter the Unit in one of the ways described earlier.
- **Effect class**
    - **Overview**
        - Represents an effect, either permanent or temporary, attached to a Unit or SkillContainer.
    - **Specifics**

        - Like Skills, Effects are ScriptableObjects, and serialized instances of them are stored in the “Assets/Resources/Effects” directory.
        - Like Skills, Units maintain links to Effects through wrappers called EffectContainers, which contain mutable information such as remaining cooldown.
        - Effects have functions that implement their impact on certain Unit Actions. These can be empty functions.
        - Units have an array of function containers that they iterate through each time a situation occurs that may be affected by Effects.
        - Effects coming from talents are given special EffectContainers that have no cooldown, and reimplement themselves each stage.
    - **EventListening Functions**
        - OnTurnStart(Unit user) : void
            - When a Unit begins it turn, It calls OnTurnStart(this) for all effects.
            - The following functions work
        - OnTurnEnd(user) : void
        - OnSkillUse(user) : void
        - OnMove(user) : void
- **AIAgent class**
    - **Overview**
        - Non player characters will need to be controlled by some form of Ai. the AIAgent class will be supplied with information about the Agent’s surroundings and queried for Commands.
    - **Specifics**
        - AIAgent has a Unit variable which points to the owner that it is making decisions for.
        - AIAgent contains a List of other Units in the game scene. At the start of the decision making process, this list is iterated through and destroyed Units(references to destroyed objects are automatically nulled) are removed. This list is then sorted by square magnitude distance from the agent.
        - AIAgents generate a list of available actions for that turn.
        - Each action has an array of possible outcomes generated based on the probability of a skill hitting the enemy.
        - The AIAgent has a general formula for evaluating a world setup. The probability-summed predicted world setup of each action is evaluated by this formula.

    - The formula is based on current resources, temperament variables and nearby allies and enemies.
    - The highest sum-valued choice is used to generate a Command to send to the GameManager, which ends the AIAgent's turn.

Version Control

- The Unity project folder is maintained in a GitHub repository.
- All members of the project use Git Bash to push and pull their changes.

Project Schedule

	Sean	James	Matt	Andrew
Week 5	Implement tile grid system	Complete UI for ActionBar	Implement main menu, write music	Help with grid system and also do a research spike on modeling
Week 6	Implement functional unit system with hard coded stubs for skills	Complete TurnOrder UI on top of the game screen	Create audio for skills/effects	Help with implementation of the unit system
Week 7	Implement skill system and ProcessCommand method for game manager	Bring sprites and environment assets into the game	Model decorative objects for level backgrounds	Work on the creation or obtaining of the models for the game
Week 8	Implement unit-effect system	Complete Skills class for the each character unit	Record and mix the music for the game	Work on character and enemy design specifically on classes and their skills
Week 9	Write eval function for enemy ai	Complete skills menu UI on pre-game Menu	Implement individual skills and talents	Work on game balance and game fairness
Week 10	Fully Implement enemy ai	Test final game and fix all last	Test abilities to make sure	Finish getting models and also

		minute bugs	teams function ideally, adjust skills as necessary	help add to the requirements doc