Western University

Faculty of Engineering
Department of Electrical and Computer Engineering
SE 2250B - Software Construction

Phase 3
Project: Space SHUMP Game

Section: 003

TA: Jose Miguel Alves

Date Submitted: April 8, 2019

Project Name: Space Wars

Team Members: Lucas Cordeiro, Zack Masciangelo, Trevor Smith

Table Of Contents

1. Power-Ups

Requirement Elicitations

The power-up feature will be a cube gameobject that falls from the enemy. This means that the enemy will drop a power-up and the power-up gameobject will fall completely vertically down the screen. The power-up falling down should also have a text letter component which specifies what type of power-up it is. Once the power-up goes off screen, it should be destroyed. As an aesthetic, the power-up gameobject should rotate as it falls. If the hero (user) collides with the power-up object, then the hero must be able to collect that power-up (destroy it) and then use that particular power-up by adding a weapon or increasing shield. The power-up feature must be dropped randomly by an enemy - not all enemies will necessarily drop a power-up.

The key components of the Power-Up feature include the power-ups, the enemies, the main, and the hero. Power-Up will be the class that contains all of the methods and fields for the characteristics associated with the power-up such as the type, colour, and bounds check. The enemy class is responsible for notifying the main when a enemy is destroyed to calculate whether a power-up will drop at that moment. The hero class is responsible for checking whether a power-up is absorbed and if so, the hero class must appropriately set the upgrade or weapon to the hero game object.

The first power-up will be of weapon type shield. If absorbed by the hero, this power-up will give the player a random shield level addition - without going above the maximum of 4 shields. For example, if the player has two levels of shield, then the power-up may add 1 or 2 levels of shield randomly.

The second power-up will be of weapon type warp. If absorbed by the hero, this power-up will allow the player to access the left side of the screen by exiting the right side. This is a classic power-up feature as seen in brick-breaker. If the hero object moves off one side off the screen, it will appear on the other side instead of being bounded by screen walls. This power-up allows for three warps and will expire after the third warp.

The next power-up will be of weapon type boost. If absorbed by the hero, this power-up will give the player a temporary speed boost to their ships movement. For instance, if the user kills an enemy and collects the boost their speed will increase by a factor of two for five seconds. This is a common power-up in most racing games.

The last power-up will be of weapon type bomb. If absorbed by the hero, this power-up will destroy the player and as a result end the game. There should be an equal chance for each of the 4 power-ups (shield, boost, warp, bomb) to be dropped.

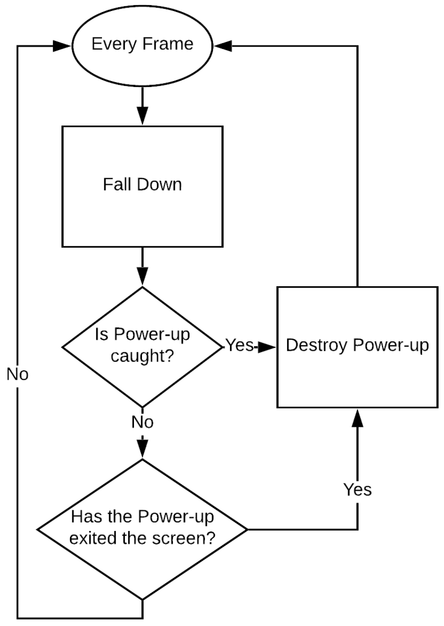
Key Components Using Object Oriented Approach:

- The power-up is an object so that if multiple power-up with different functions want to be implemented it can just be added into this class via the Main class
- The enemies are key components because whenever a enemy is destroyed they have a power-up chance variable deciding whether a power-up must be dropped which is accomplished by the power-up class

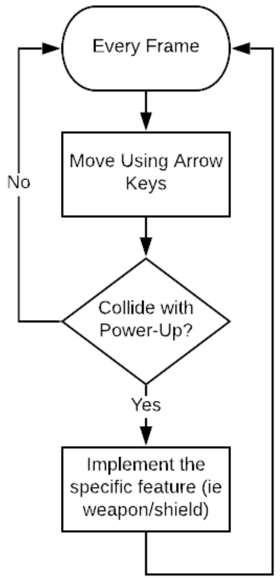
- The hero is an object that must change when it encounters a power-up. When the hero picks up a power-up it then calls on the power-up class to check which power up it is and then assigns the change in the hero class

Analysis

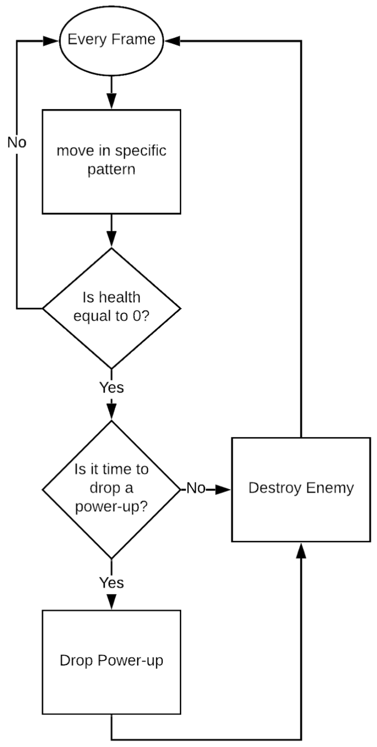
Power-up Flowchart

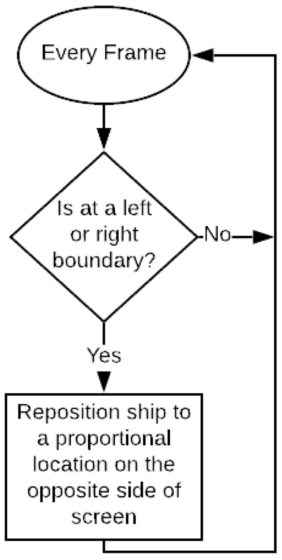


Hero Flowchart



Enemy




Every Frame
Is at a left or right boundary?
No
Yes
Reposition ship to a proportional location on the opposite side of screen

A flow chart for the warp power-up has been added to help technically describe the implementation of this power-up. The shield power-up does need a flowchart to represent the situation of how the shield is implemented. The description is stated above in requirement elicitations. Inside the absorbPowerUp() function, a switch case will be added for shield and random shield level will be added. The boost power-up also does not need a flow chart to represent the situation because it is simply implemented by adding a certain amount of speed for a certain period of time.

System Design

- Exceptional situation where two enemies killed at once: in this case, the ship-destroyed function will be called simultaneously from the main script attached to the main camera and this allows for multiple enemies to be killed at same time and for multiple power-ups spawned at the same time.
- Exceptional situation where multiple caught at once: in this case all power-ups that are absorbed will be applied simultaneously meaning that, if a boost and warp power-up are absorbed at the same time, the players speed will be increase and they will be able to warp.
- Exceptional situation where two boost are absorbed in a row: the second boost will not increase speed again but will increase duration of boost.
- Exceptional situation while gathering multiple warps: the max amount of warps is three. This causes following warps to increase the user's number of available warps to three.
- Exceptional situation where user has zero warps: warp UI text disappears.

Object Design

- Reuse: The power-up artwork from Jeremy Gibson Bond, Introduction to Game Design, prototyping, and Development: Chapter 31, Prototype 3.5: SPACE SHMUP PLUS will be used to create our gameobject prefab for Power-Up. The textbook will also be reused to implement the events that occur when the power-up is dropped and caught by the means of methods and variables such as the absorbedby() function. The particular ways in which the hero responds, such as the weapons or upgrades each variable represents, will not be reused.

2. Level Progression

Requirement Elicitations

Level Progression will keep the game challenging but also allowing you to learn and master your skills. As the player continues to play the game certain predetermined checkpoints will be reached (Every 50 points). These checkpoints cause a “Boss” to spawn and the next level to commence. A level commences by displaying the current level, the boss's health and fire rate for that level. Next the spawn rate of the normal enemies are changed to 0 meaning that no other enemies will spawn while the boss is spawned.

The next step in the level progression is the the spawning of the boss and how it escalates as the player progresses. The boss is spawn after a slight delay and is instantiated with a base health and fire rate that is multiplied by the current level. This will cause the boss to become progressively more challenging as the player advances. The Boss then proceeds to move around to random locations and shoot in a three projectile burst straight down.

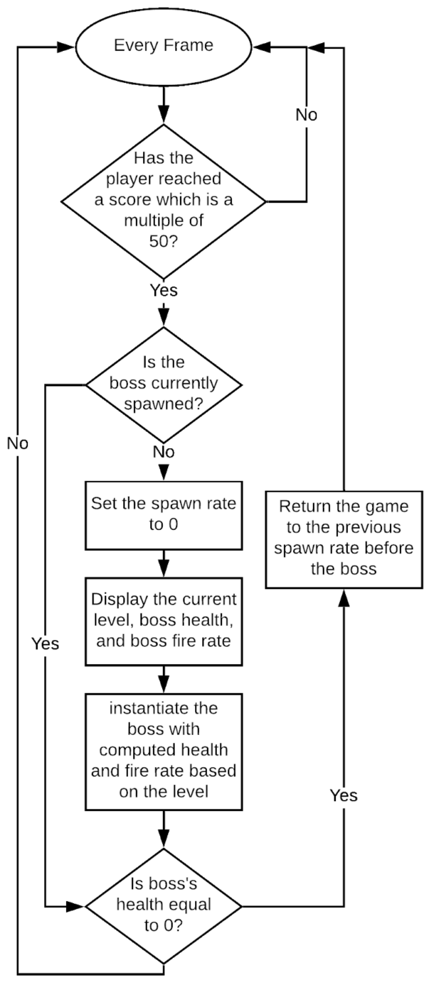
Once the boss’s health reaches 0 the boss is destroyed. This causes the level to be over calling the levelOver() function. This will take the spawn rate and return it to its previous value before the boss level occurred. This process will continue and create a progressive cycle that gets harder and harder each time.

Key Components Using Object Oriented Approach:

- The first object was the BossLevel Script. This script is in charge of the boss spawning and ending level operations this is its own game object because it can be reused for other bosses
- BossEnemy was inherited from Enemy but had slightly different features so it required its own prefab and script
- LevelManager kept track of the current level the user is on and the information that needs to be displayed. This is its own object because other counters or level types can reuse this script

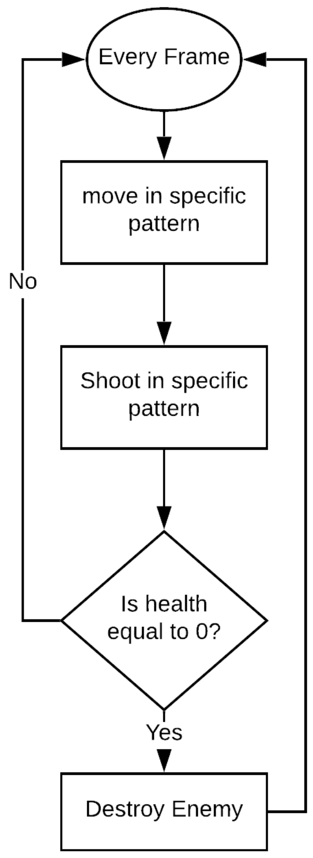
Analysis

Level Manager



Every Frame
Has the player reached a score which is a multiple of 50?
No
Yes
Is the boss currently spawned?
No
Yes
No
Set the spawn rate to 0
Display the current level, boss health, and boss fire rate
instantiate the boss with computed health and fire rate based on the level
Is boss's health equal to 0?
Yes
Return the game to the previous spawn rate before the boss

Boss Enemy



System Design

- Exceptional where the Boss might hit the bounds: This is dealt with in the Boss's movement.The Move() function allows it to only move inside the bounds.
- One boundary use case is when instantiating the Enemy Boss. When it is time the boss is instantiated dynamically and all components of it are dynamic like the Weapon so all variables must be set dynamically.

Object Design

- Reuse: The enemy scripts have been reused in order to lay out the groundwork for the Boss Enemy. This means that functions such as OnCollisionEnter() and the Awake() functions will be reused. In addition, The framework for the Weapon prefab and Weapon class were used in order to get the boss enemy to shoot . this was done by adding a weapon prefab to the boss and adding a new weapon type called enemy. The level manager script took a lot of inspiration from the Scoremanager in the Prospector Solitaire game and the Score manager in our game currently.

3. Voice Recognition

Requirement Elicitations

The voice recognition feature is an exciting, creative part of this project. This feature will be able to recognize a set of predetermined phrases and the game will then respond appropriately. The first phrase that will be recognized is “more enemies” or “harder”. This phrase allows for a better adjustment to flow for the player. By saying this phrase, the game will increase the spawn rate of enemies by 0.75 and therefore, make more enemies appear faster. If the player is finding the game too hard, it may say “less enemies” or “fewer enemies” or “easier” and the spawn rate of enemies will decrease by 0.75. This will ensure that the player does not get bored or frustrated from difficulty.

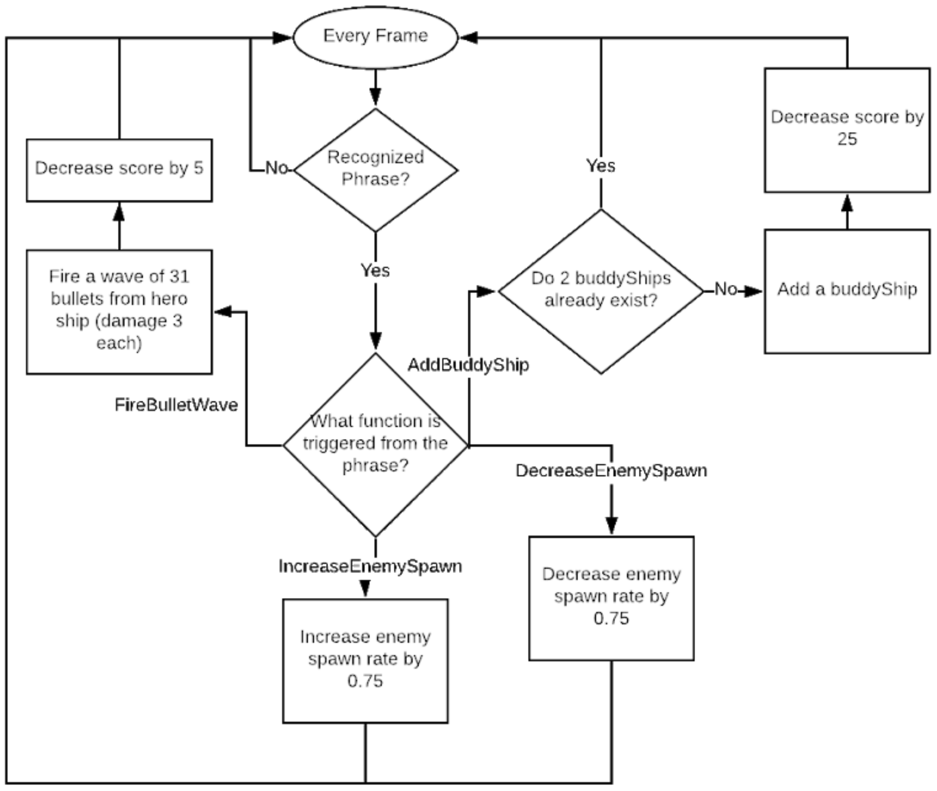
The voice recognition feature will also be able to recognize the phrase “Jose help” which will fire a wave of 31 bullets from the hero ship. The bullets will shoot from the hero ship in an equally distributed angle upwards. For example, one will shoot straight upwards and the remainder will shoot at various angles to the left and the right side. The bullets should be of colour orange. Additionally, each time the “Jose help” or “Help” phrase is used, it should cost 5 points. In the case it is used, the score should therefore decrease by 5. If the score is not equal to or greater than 5, then this component of the voice recognition should not be available and does not take action when said.

The final component of the voice recognition feature is the ability to purchase an additional buddy ship. By saying “Buy Friend” or “Buy Buddy Ship” or simply “Buddy Ship”, the score will decrease by 25 and a new buddy ship will appear on the screen to help the hero. This call will not be available if there are already two buddy ships on the screen or if the accumulated score is less than 25. The new buddy ship should act, perform, and move the same way as the ones that are initially started in each new game.

Key Components Using Object Oriented Approach:

- Voice recognition will be attached to main camera and whenever phrase is said, a particular function will be activated
- Hero component will be affected for when “Jose Help” is said because its weapon type which change for the instantaneous moment for when the 31 bullets are shot
- In the same fashion that enemies are affected by other bullets, the definition of the 31 bullet weapon in the weapon script should damage the enemies as well
- “More/Less Enemies” will edit the same attribute in the main script to maintain standard of object oriented approach
- Creating a new buddy ship will use Score Manager to decrease score, buddy script and buddy ship prefab to instantiate a new buddy ship
- Sound Manager will also be used for each of the sounds with respect to voice recognition. This is detailed further in the “Sound Effects” section of this documentation

Analysis


Every Frame
Recognized Phrase?
No
Yes
Decrease score by 5
Fire a wave of 31 bullets from hero ship (damage 3 each)
FireBulletWave
What function is triggered from the phrase?
AddBuddyShip
Do 2 buddyShips already exist?
Yes
No
Add a buddyShip
Decrease score by 25
DecreaseEnemySpawn
Decrease enemy spawn rate by 0.75
IncreaseEnemySpawn
Increase enemy spawn rate by 0.75

System Design

- Exceptional case when multiple phrases are said at the same time (could be by many users speaking at once). In this case, the built-in phrase recognition function can only pick up on one phrase at a time. Two phrases, therefore, shall not be processed or said at once.
- Exceptional case when the phrase is not recognized. Since this function is reused and part of a hardware component, there is nothing that can be done other than for the user to try to say the phrase again. The microphone, in general, is very effective at picking up on phrases.

Object Design

- Reuse: The UnityEngine.Windows.Speech library was reused for this voice recognition feature. This library allowed for the phrases to be recognized through voice commands and then through the use of a dictionary, the phrases were appropriately matched with a specific function/action.

4. Sound Effects

Requirement Elicitations

The sound effect feature helps the user audibly recognize events during the game and helps keep the user engaged.

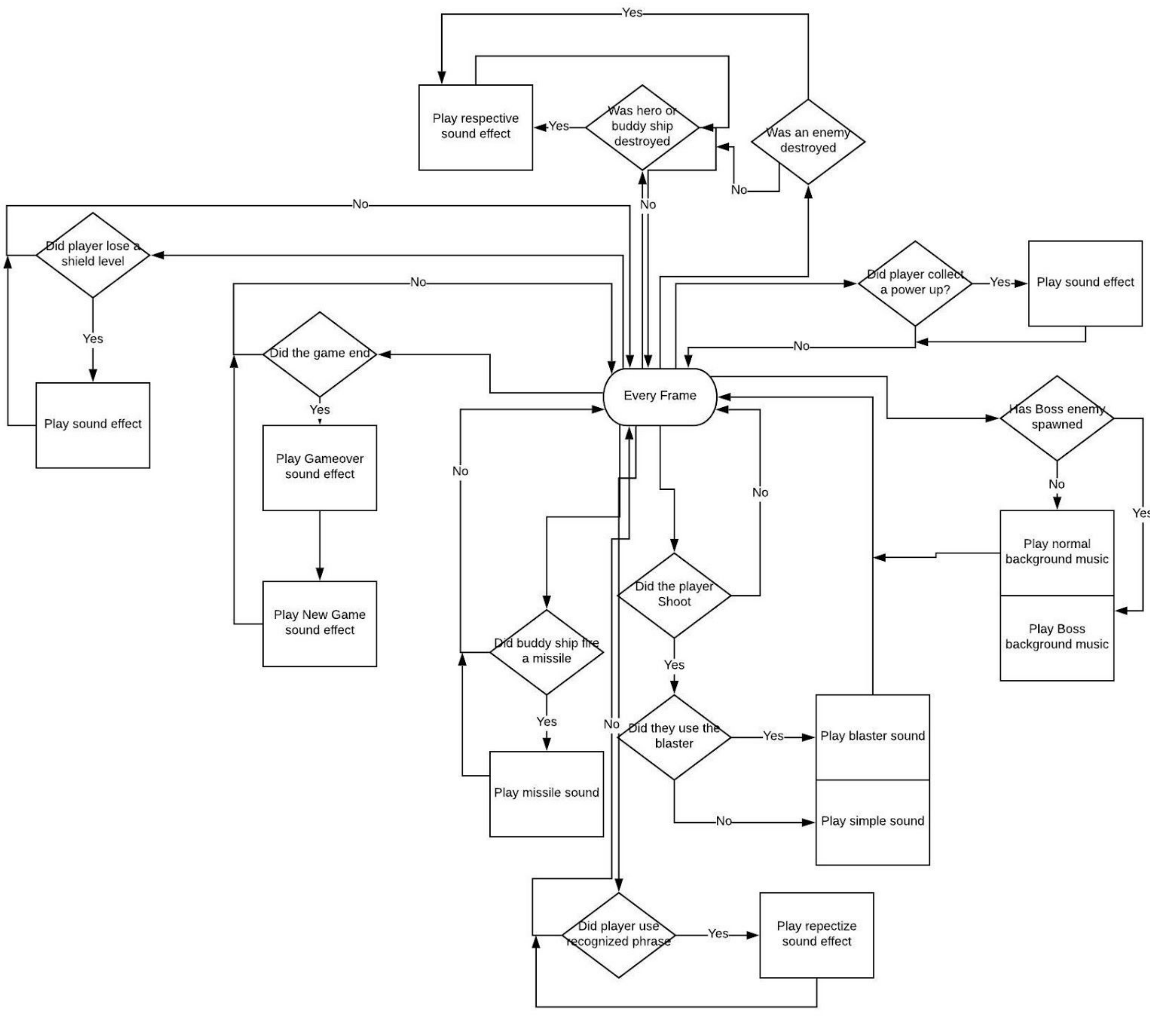
There should be a sound implemented for the following situations:

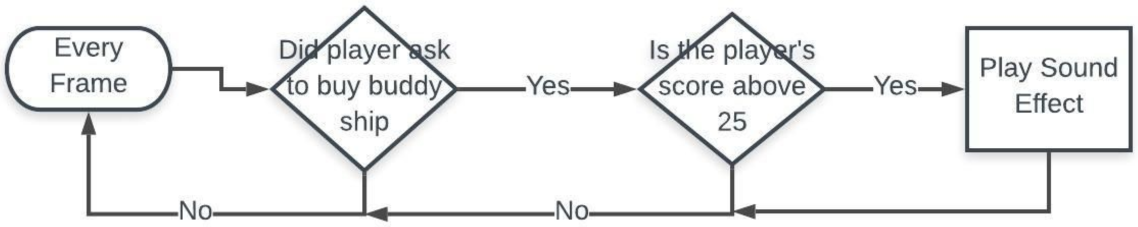
1. Hero shooting simple weapon type
2. Hero shooting blaster weapon type
3. BuddyShip shooting homing missile weapon type
4. “New Game” voice at the start of each game
5. “Game Over” voice at end of each game
6. “Boost” voice when collecting boost power-up
7. “Shield Up” voice when collecting shield power-up
8. “Warp” voice when collecting warp power-up
9. “Vroom” sound when any buddyship or hero proceeds in the action of warping
10. Background electronic-space music
11. Background music switches when boss enemy approaches and then switches back to regular music when destroyed
12. Boss Enemy each time it shoots its weapon type
13. Glass breaking sound when an enemy or enemy bullet collides with hero’s shield
14. “More Enemies” voice when any phrase to increase spawn rate is said
15. “Less Enemies” voice when any phrase to decrease spawn rate is said
16. “Jose is Here” voice when any phrase to trigger wave of 31 bullets is said
17. “Jose Unavailable” voice when “Jose Help” or “Help” is said and score is less than 5
18. Explosion sound when buddyShip or hero is destroyed
19. Different Explosion sound when enemy is destroyed
20. “Buddy Unavailable” voice when score is less than 25 or two buddy ships exist
21. “Buddy Added” voice when another buddy ship is instantiated via “Buy Buddy Ship”

Key Components Using Object Oriented Approach:

- Two scripts will be made: one that defines the class Sound and one to manage them by creating Play and Stop functions.
- Sound Manager will be attached to an empty game object and whenever an event occurs in the game, a particular sound will be activated.
- The functions in Sound Manager will then be called from other scripts to be played when an event occurs.

Analysis


Every Frame
Was hero or buddy ship destroyed
Play respective sound effect
Was an enemy destroyed
Did player lose a shield level
Play sound effect
Did player collect a power up?
Play sound effect
Did the game end
Play Gameover sound effect
Play New Game sound effect
Has Boss enemy spawned
Play normal background music
Play Boss background music
Did the player Shoot
Did they use the blaster
Play blaster sound
Play simple sound
Did buddy ship fire a missile
Play missile sound
Did player use recognized phrase
Play repectize sound effect
Yes
No


Every Frame
Did player ask to buy buddy ship
Yes
Is the player's score above 25
Yes
Play Sound Effect
No
No

System Design

- An exceptional case when multiple power-ups are collected at once. The sound effects will execute at the exact same time instead or a poorly structured queue-style sound execution.
- An exceptional case when “more enemies” and a power-up sound are said at the same time. The sound effects will execute at the exact same time instead or a poorly structured queue-style sound execution.
- In can be seen that in all cases of combination of sound effects, when two play at once, they will play on top of one another.

Object Design

- Reuse: Sound effects were reused from freesound.org
- Sound effects that were created by our team: “New Game” voice, “Game Over” voice, “Boost” voice, “Shield Up” voice, “Warp” voice, “More Enemies”, “Less Enemies”, “Jose is Here”, “Jose Unavailable”. These sounds were not reused, but rather created specifically for this Phase 3 project.

5. Unlock Skins (Change Ship Colour)

Requirement Elicitations

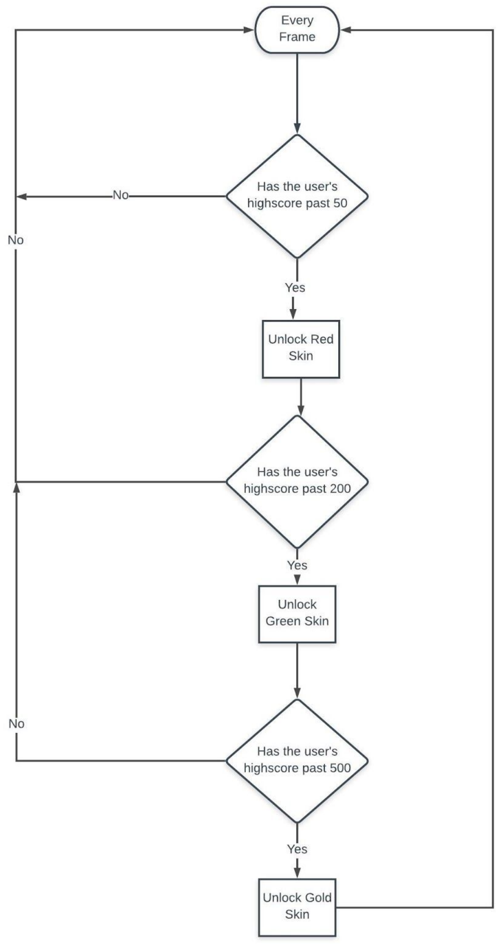
The unlocking skins feature enables players to feel a sense of accomplishment besides from beating their highscore. It also allows multiple levels of players to set goals to unlock their next skin while playing the game. Each time ‘x’ button is pressed, the ship will attempt to change to a new colour. There are a predetermined set of colours for the ship to be: white/normal, red, green, and gold. A component of this feature is that the number of colours “unlocked” or available to the user is dependent on the player’s highscore. Red is unlocked when the user has a highscore of 50, green is unlocked when the user has a highscore of 200, and gold is unlocked when the user has a highscore of 500. Using skins increases the users ship speed depending on the rarity of the skin. The greater highscore needed to get the skin, the more speed bonus the user will get while using that skin.

Key Components Using Object Oriented Approach:

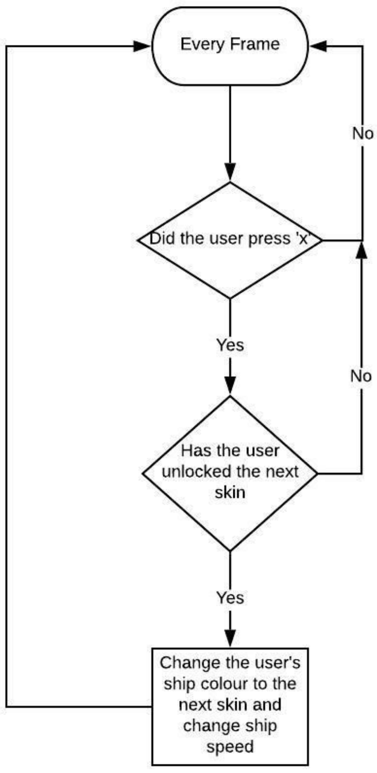
- In the hero class two functions are added: one that check the players highscore to unlock a skin and the other to listen for when the user wants to change skin and thus speed.
- These functions are then called in the update function of hero.

Analysis

Unlocking Skins


Every Frame
Has the user's highscore past 50
No
No
Yes
Unlock Red Skin
Has the user's highscore past 200
Yes
Unlock Green Skin
No
Has the user's highscore past 500
Yes
Unlock Gold Skin

Changing Skins


Every Frame
No
Did the user press 'x'
Yes
No
Has the user unlocked the next skin
Yes
Change the user's ship colour to the next skin and change ship speed

System Design

- Exceptional case where user tries to change skin when currently using a boost power up: the system has been intentionally set up so that the user will be unable to change skin until the boost power up wairs off.
- Exceptional case where the user’s score passes the checkpoint to unlock their next skin: the system has been intentionally set up so that once the game restarts and the user’s highscore is changed the user will be able to change to their new skin.

Object Design

- Reuse: since skins are based off of the user’s highscore the static public ScoreManager will be reused to access the user’s highscore.

6. Buddy Ships

Requirement Elicitations

There will be two buddy ships that must appear on each side of the hero ship at the beginning of each game (7 x-position units on either side). They should be gold in colour and have a weapon gameobject as a component. While the hero can only warp depending if it has warps remaining from a power-up collection, the buddyShip must be able to warp an infinite amount of times so that the hero can move to the edge of the screen whenever. When a friendly buddy warps, it should not affect the number of warps for the hero ship. Additionally, when the buddyShip warps back to the original position, it should be the same distance away from the hero ship.

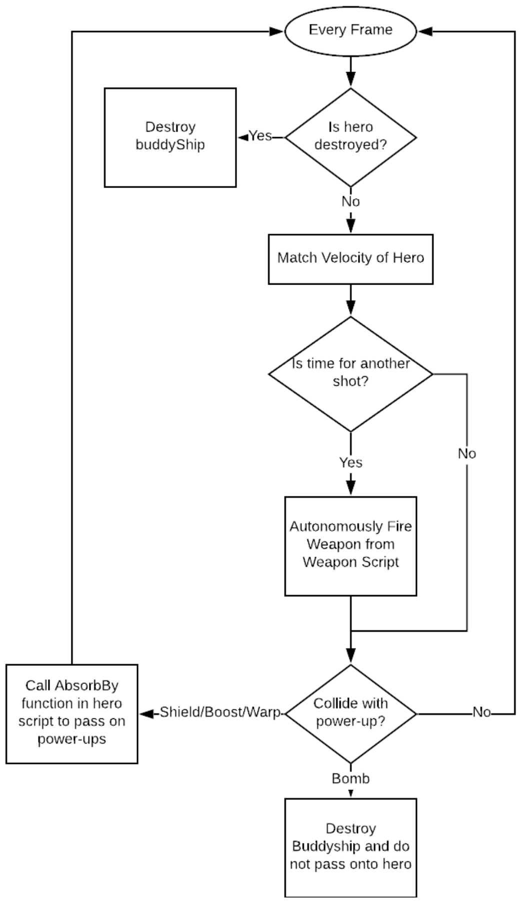
These buddyShips should always move at the same velocity as the hero (even when the hero has boost or changes colour). When the hero is at a boundary edge and is not moving, the buddy ships should also not be moving. Furthermore, when a buddy ship collides with a power-up, it should collect the power-up in a similar way the hero does. When a buddy ship specifically collects a shield, boost, or warp, it should affect the hero the same way as if the hero picked up that power-up itself. But, if the power-up collected by the buddyship is a bomb, then it should only be destroyed and should not destroy the hero.

Depending on the high score the player has achieved, the buddyShips will shoot at different delay fire rates. The greater the high score, the lower the delay in fire rate. This allows users who are already skilled to progress faster in the game and ensure appropriate flow. Each buddyShip will only have one health. So, if the buddyShip collides with another enemy, or an enemy projectile, then the buddyShip should be destroyed.

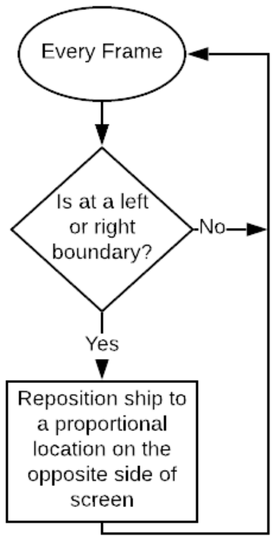
Key Components Using Object Oriented Approach:

- Will access speed velocity from the hero gameObject and match its speed
- If absorbs power-ups, the hero script should be executed so that the power-up affects the hero as well
- Bounds check script will be used with object oriented approach in which the script will be attached to each buddy ship

Analysis



Every Frame
Is hero destroyed?
Yes
Destroy buddyShip
No
Match Velocity of Hero
Is time for another shot?
Yes
No
Autonomously Fire Weapon from Weapon Script
Collide with power-up?
Shield/Boost/Warp
Call AbsorbBy function in hero script to pass on power-ups
No
Bomb
Destroy Buddyship and do not pass onto hero



Every Frame
Is at a left or right boundary?
No
Yes
Reposition ship to a proportional location on the opposite side of screen

To improve organization and readability, the flowchart of the buddyShip has been broken up into two sections. The flowchart on the left is the main guideline for the buddyShip while the smaller flowchart on the right is responsible solely for the warping of the buddyShip.

System Design

- One boundary use case is when instantiating the buddyShips. There will be two buddyShips that start in the scene at the beginning of each game. No buddyShips are instantiated dynamically.
- An exceptional use case is when the hero ship collects a boost and its speed increases. This is dealt with by constantly ensuring the buddyShips' speeds are the same as the hero ship's speed in each update() call.

Object Design

- Reuse: The buddyship will implement similar features to the hero in the way that it rotates and moves. While this would suggest an inheritance class should be used to combine similar functions of both the buddyship and hero, each function is, indeed, differently so it would not be effective to override each function. While some lines of codes are reused from the hero script, since no two functions are the same, the use of inheritance was decided against.
- The bounds check script will be reused for these buddy ships.

7. Homing Missile

Requirement Elicitations

The homing missile is the seventh additional feature that will be implemented in the space shooting game. This type of weapon will be green in colour, cause one damage on hit, and will have a standard firing delay right of 1. Additionally, the missile will shoot with a velocity of 20. The projectile of this homing missile will look different from all other weapon types. This feature will have a missile prefab from the asset store.

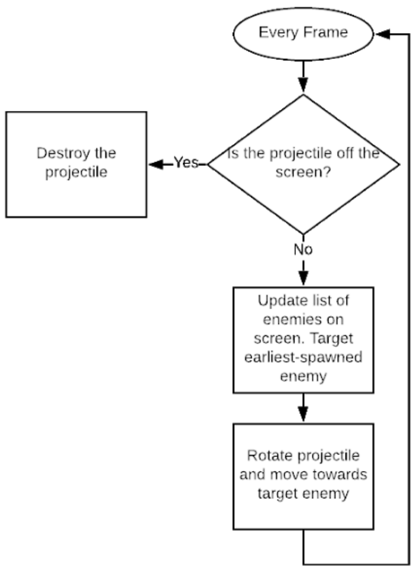
The main component of this feature is the projectile's ability to know the locations of all enemies on the screen and target the earliest-spawned enemy. The projectile will then actively rotate to "seek out" and constantly move towards the target enemy. If the target enemy is destroyed before the projectile has reached it, the projectile should then "seek out" the next enemy in the list of target enemies. If there are no other enemies on the screen, the missile should continue moving in the direction it was moving previously. If the projectile goes off the screen, the object should be destroyed.

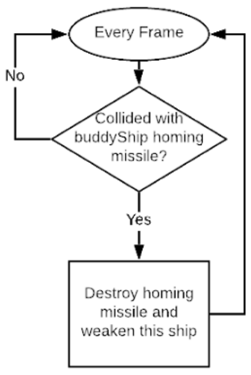
This weapon will only be implemented by the buddyShips. And, this will be the only weapon that the buddyShips use.

Key Components Using Object Oriented Approach:

- The missiles will be shot from the buddy ship game objects so these will be one of the components involved in this feature
- Another component is the missile itself and a script attached to its game object will control how it moves

Analysis


Every Frame
Destroy the projectile
Yes
Is the projectile off the screen?
No
Update list of enemies on screen. Target earliest-spawned enemy
Rotate projectile and move towards target enemy


Every Frame
No
Collided with buddyShip homing missile?
Yes
Destroy homing missile and weaken this ship

To improve organization and readability, the flowchart of the components involved with homing missiles has been broken up into two sections. The flowchart on the left is how the missile projectile will act while the smaller flowchart on the right is responsible for determining how enemies will act when colliding with the homing missile. For the enemies to detect the homing missile, the projectiles will be tagged appropriately in Unity.

System Design

- An exceptional use case which must be handled for the homing missile is when the target enemy is destroyed. In this case, the projectile must know when the enemy is destroyed and to avoid runtime errors, the projectile should no longer access this error. Runtime errors will occur if trying to access a destroyed gameobject so this exceptional case is dealt with by using an if-statement check to ensure the target enemy is not null before trying to access it.
- The second exceptional use case that exists with this feature is if it is tracking an enemy but it goes out of bounds during the “tracking”. In this situation, the “tracking” should halt and the projectile should be destroyed.
- The third and final exceptional use case is when the missile is shot and the buddyShip from which the projectile is shot is destroyed. In the case, the missile should not be destroyed when the buddyShip is destroyed - the projectile should continue seeking out enemies as in a normal case.

Object Design

- Reuse: Missiles from asset store under the title “A set of original PBR space missiles”. The hook missile is the particular missile that was used from this set of prefabs from the asset store. The hook missile is then coloured green as outlined in the requirement elicitations.
- The bounds check script will be reused for this projectile missile.

8. Main Menu

Requirement Elicitations

There should be three components and three buttons involved with the main menu. There should be an instruction page which has the instructions of how to play the game and the details of what each power-up will do. The instructions that should be displayed can be found in the attached pdf of the OWL submission titled “Space Wars Instructions”. There should be a back button which allows the user to go back to the main menu from these instructions. The second component with the main menu should be a start button. This button will allow the user to begin playing the game. The third component should be an exit button and when pressed, this button will exit and stop running the game.

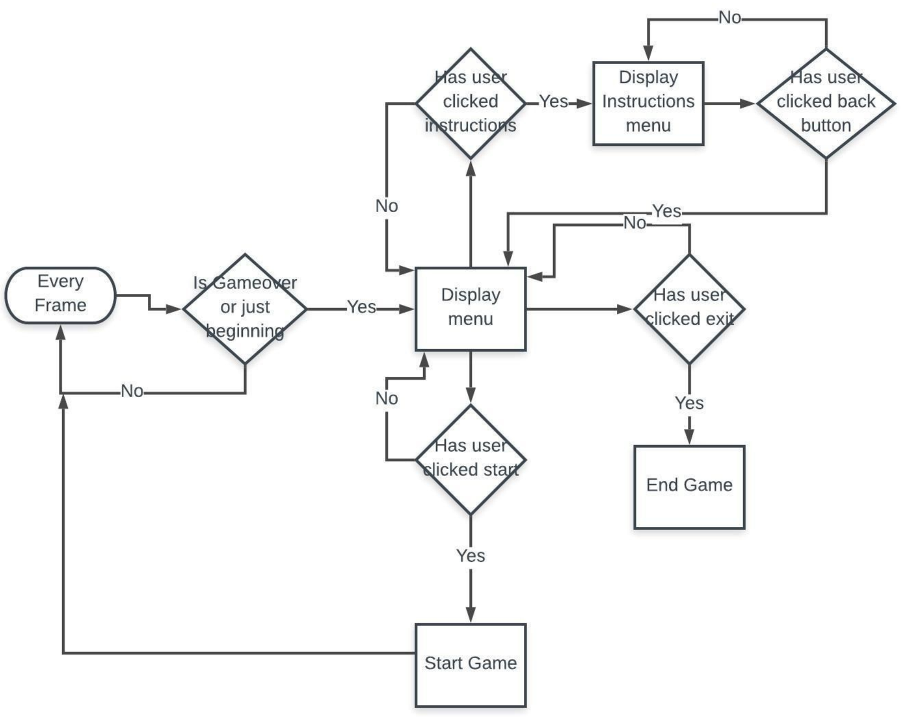
It should be possible for the user to navigate the menu with only use the keyboard. Up and down arrows should enable the user to highlight each button and then the enter button enables the user to “click” the button.

The main menu will show up at the very beginning of the game. When the player dies, the main menu will pop up again and allow the user to make another decision of whether to view instructions, exit, or play another game.

Key Components Using Object Oriented Approach:

- The first object component to consider is that this will be another scene so a new scene must be created
- Scene must then switch from the menu scene to _scene_0 when the appropriate buttons are clicked
- Overall, considering these two components will enable this feature to follow the object oriented approach. Additionally, each button will be its own object as well

Analysis



Every Frame
Is Gameover or just beginning
Yes
No
Display menu
Has user clicked instructions
No
Yes
Display Instructions menu
Has user clicked back button
No
Yes
Has user clicked exit
No
Yes
End Game
Has user clicked start
No
Yes
Start Game

System Design

- Boundary use case will be when the game starts up. As soon as the game starts up, this menu feature should be displayed
- A second boundary use case is when shutting down. The main menu button “exit” will cause the program to shutdown

Object Design

- Reuse: This main menu will reuse the GUI menu from the unity asset store labelled “UnitySampleMenu”.