Deep Space Emissary

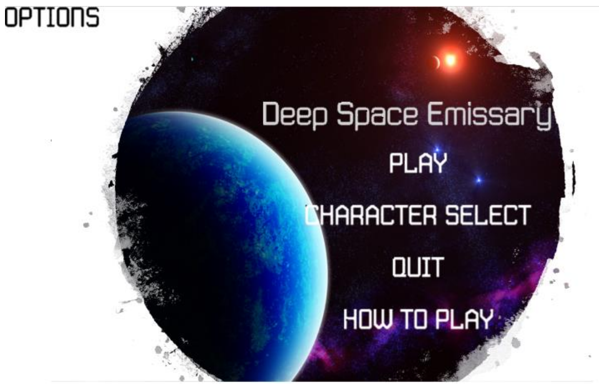
CI328 – Single Player Game

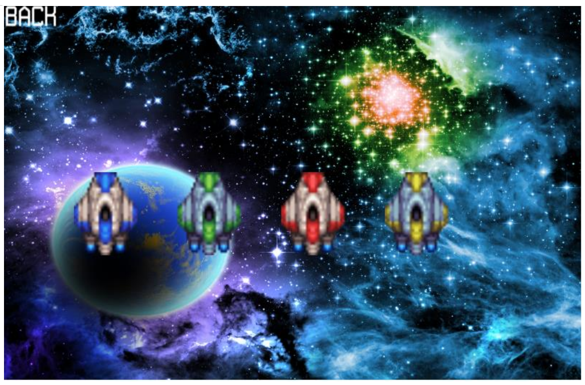
Doryan Sadi – 17850125

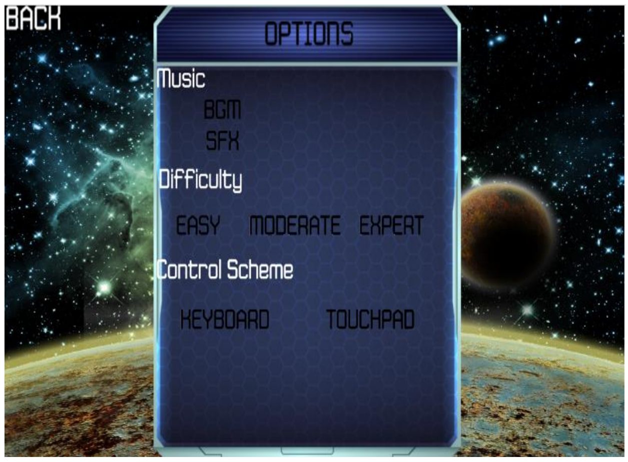
Game Summary:

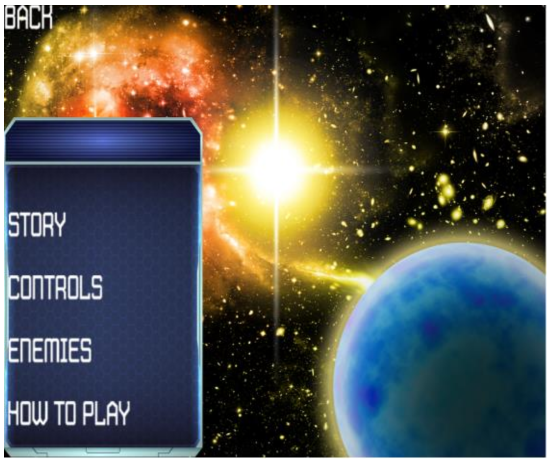
Deep Space Emissary is a game created based on retro/"old school, shoot 'em up's" (Also known as 'Shmups') and Space Invaders. Similar mechanics are present. You control a ship which can only move left or right and fire a bullet to kill an enemy. Each enemy kill will result in your score increasing, each kill is worth ten points. You must attempt to destroy all the enemies before they destroy you to win. Upon win or defeat the option to restart your game will be given to which you can retry to improve your score.

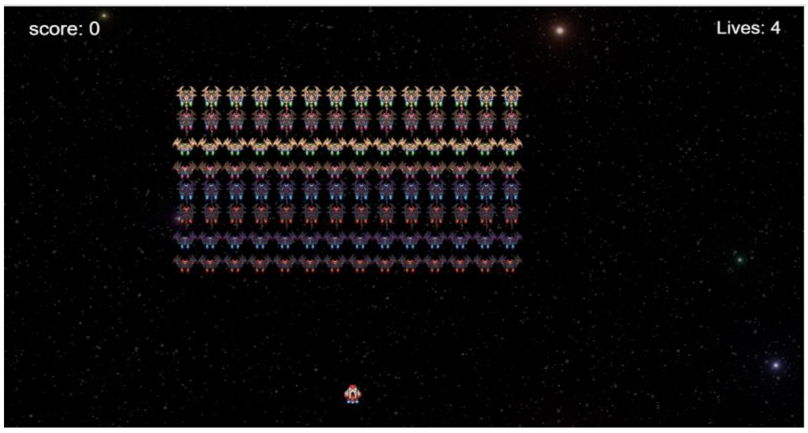
Storyboard:











The storyboard shows the progression of the gameplay for Deep Space Emissary. First image depicts the Main Menu scene, following picture represents the character select screen. Third and fourth picture is the "Options" and "How to Play" scenes. Final scene shows the gameplay.

Implementation Specification:

During the creation stage of my game, my code was organised into different .js files depending on the code's role. For example, all code created for creating the gameplay was implemented there while other code was developed on a different .js file. This procedure helped with code sophistication as less code reduced disorganisation. All .js files would begin with a variable which will store all functions and methods for that scene as a class, this class is then called at the end by a variable called "deepSpace.scenes" adding the scene into the scenes group.

The layout of each .js file will contain a "preload", "create" and "update" function. The preload function will always be tasked with loading in images, pictures and sprite sheets. The create function contains code for all technical aspects, such as: methods, functions and algorithms. The update function is used to update the game world. I preferred the use of methods during the creation process as it helped me create multiple.

I was able to implement a "space invaders" type game by creating classes and groups to group together sprites and assign a variable allowing me to control sprites simultaneously. An example being the creation of my enemy sprites. I began by creating a "createBiomech" method which contains the code needed to load the sprites onto the game world. I then called the method in the create function. By adding the enemy sprites as a group, I used the group properties to duplicate my sprites in a line, I repeated this process with my other enemy sprites.

Creating methods also allowed me to control when I wanted events to occur. This helped greatly when creating the enemy's animations and tweening. I created two new methods, one called "createBiomechAnimations" the other called "createBiomechTweens". By using the "this. anims.create" and utilising the parameters given I created animations for the aliens by first loading the enemies as a sprite sheet and creating an animation by using the frames, setting the frame rate and repeating the animation. Tweening was done in most part similar to animation in which I used the function "this.tweens.createTimeline()" to create multiple groups of tweens which will move the sprites along the x and y axis. Another algorithm I created was one for collision detection. By enabling all my sprites was physics, I was able to use the collision detector to destroy a sprite on event of a collision.

To do so, I first created a group of bullets enabled with physics that when key "W" is pressed a "bullet" is fired. I used a mixture of group and class to create functions and set variables that is referenced to by the group associated to it. I also used the update function to update the players movement when the Keycodes have been pressed.

Research:

Creating a "Shmup" required a lot of research. Most research was spent reading resources from Phaser.io as they have a library suited to help for many different aspects of coding. When reviewing examples of "Space Invaders" games and Shmups, many similar code structures were present helping me narrow down which approach I should take. This influenced by game as the design is made reminiscent of 90's Shmups games using sci-fi imagery such as planets, stars, low-fi music etc.

In terms of implementation, there were many ways of implementation I could have use to build my game, however, most of them failed. One method was creating a separate .js file which will contain entities for the player, enemies and bullets. These entities will contain properties for the sprites which will be called in the game file. This did not work due to confusion as properties for my sprites were already implemented causing multiple errors.

Critical Review:

While reviewing the result of my games development, I believe there are three reasons as to why my games design and implementation excels and where it can be improved. I believe the design of my application is very impressive as the use of high-resolution pictures and sprites is very aesthetically pleasing to players. Another improvement in terms of design would be the musical choices for my game's development fits the tone and atmosphere intended. A good aspect in terms of implementation would be the methods system I used.

This made it easier to group tasks individually. By creating methods, I was also able to avoid repetitive code decreasing the amount of errors or loading time. My code could also be improved in three areas. Code simplification, my code could be re-written to have lesser repetitive code. One way to go about this would to learn Phaser 3's documentation more thoroughly, although in conjunction to that solution, Phaser 3's implementation could also be polished in order to help those learning to understand.

Two other aspects my application could improve on is my use of plugins, in future I would mostly abstain from using plugins as they are very hard to implement and can lead to multiple errors. One final aspect of improvement is fixing some broken code. As previously mentioned through the gradual increase in use of Phaser 3 labs and more experience better understanding will result in fixing broken code to create a more complete and concise game.

Assets:

Assets I used in the game are space themed assets featuring planets to further immerse the player with the atmosphere of the game. All assets including the music is used to create a sci-fi space theme. One music choice, in-particular is the How to Play menu's background music called "low-fi". This background theme gives an eerie feeling while also having an upbeat tempo. This is good for setting the mood and increasing the action for players.

References:

"Chaingun, Pistol, Rifle, Shotgun Shots" by Michel Baradari licensed CC-BY 3.0, GPL 2.0 or GPL 3.0:
https://opengameart.org/content/chaingun-pistol-rifle-shotgun-shots

"Shmup Ships" by surt licensed CC-BY 3.0, GPL 2.0 or GPL 3.0:
https://opengameart.org/content/shmup-ships

"Solar System" by ShinyOgre licensed CC-BY 3.0, GPL 2.0 or GPL 3.0:
https://opengameart.org/content/solar-system

"Space Backgrounds" by surt licensed CC-BY 3.0, GPL 2.0 or GPL 3.0:
https://opengameart.org/content/space-backgrounds-3

"Starfield Alpha" by surt licensed CC-BY 3.0, GPL 2.0 or GPL 3.0:
https://opengameart.org/content/starfield-alpha-4k

"B&W Ornamental Cursor" by qubodup licensed CC-BY 3.0, GPL 2.0 or GPL 3.0:
https://opengameart.org/content/bw-ornamental-cursor-19x19

"low-fi" by surt licensed CC-BY 3.0, GPL 2.0 or GPL 3.0:

https://opengameart.org/content/low-fi

"another-space-background-track" by yd licensed CC-BY 3.0, GPL 2.0 or GPL 3.0:

https://opengameart.org/content/another-space-background-track

"Window-Resized" by craftpix.net licensed CraftPix:

https://craftpix.net/freebies/free-space-shooter-game-gui/

"Phaser3":

https://labs.phaser.io/

Phaser 3 was used in order to help develop code, free to use under MIT License.

All other images used were created by me.

Video:

Video showing gameplay.

URL: https://www.youtube.com/watch?v=irax6vzNK1s

GitHub: https://github.com/CaptainJush/Deep-Space-Emissary