**Task 7: Design a Classic Game – Space Invaders**

Description of the Game

Gameplay in Space Invaders is relatively simple. The player controls a small ship that can only move laterally across the bottom of the screen and fires vertically. 5 rows of 11 aliens each advance slowly from one side of the screen to the other, dropping down one space and reversing direction when they reach either side. The player’s task is to acquire points by eliminating enemies and to destroy all the aliens before they reach the bottom of the screen and complete their “invasion.” As aliens are destroyed, the speed of the remaining enemies increase, as does the tempo of the music. Once all the enemies are destroyed, the wave resets and the difficulty increases.

The Invaders constantly shoot back at the player as they advance from side to side across the screen. To help avoid their attacks, the player can hide behind a few destructible barriers or “bunkers” near the bottom of the screen (four in the original version). Occasionally a “mystery ship” will appear near the top of the screen and move quickly from one side to the other. Destroying it rewards the player with a sizeable point bonus.

Graphical user interface

Description automatically generated with low confidence

Programming Patterns

Using **command pattern** on the bullet object. The bullet object will create a bullet, check if it’s a player or alien spawning the bullet then update the movement based on its type, if it is the player it will move forward, if the enemy it will move backwards on the Y-axis.

A **factory programming pattern** could be used to create the other game objects in the game e.g. player, aliens and bullet. This helps to avoid the problem of complex object instantiation by keeping it all in a single place rather than scattering it around in the code.

**Flyweight programming pattern** could be used to store the shared aspects of the aliens (textures and animations) away from the individual aspects (position and isAlive). More efficient as you can reuse the shared aspects instead of creating them for 55 aliens.

Data Structures

For all the enemy aliens I would store them in a double linked list. I believe this would be beneficial to remove individual invaders from the game. When the player bullet collides with an invader you could call the remove function for the double linked list and pass in that invader to be removed.

A hash map is also used when implementing flyweight programming patterns so that might be a better way to implement the enemy system rather than storing all the aliens in a double linked list. Then use a remove function to remove the alien that is collided by a player bullet.

Algorithms

**Bullet Spawning** – when the player presses space bar the bullet spawns and moves forward with speed multiplied by delta time, also has a small cooldown in between shooting. For the enemy the bullet will spawn randomly at one of the aliens at the front of a column then move backwards from the alien at speed multiplied by delta time.

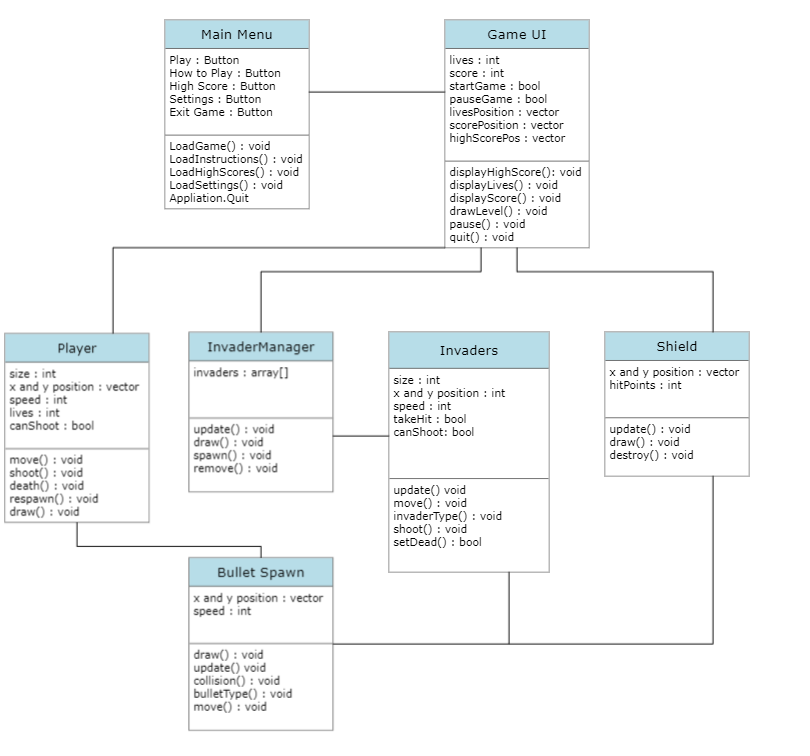
**Bullet Collision** – When a player bullet collides with an enemy the remove function will be called to remove that alien from the double linked list. When the enemy bullet hits the player the game pauses, checks if player has any lives, the player is destroyed and loads another player then resumes game. If the player is out of lives, it is game over. If the player or enemy bullet hits the shields, then a part of the shield is destroyed which means the player has less cover from incoming bullets.

**Enemy Movement** – All enemy aliens will have an x and y position and sprite that is stored in the double linked list. Will have to update the position with speed multiplied by delta time for each alien as they move side to side and down towards the player. It should get easier to update all enemy movement as the game goes on as there will be less enemies once the player starts shooting them.

**Player Movement** – The player will need a speed variable, X and Y position. The X position will need to be updated depending on if the right or left arrow key is pressed and bound the movement to the edges of the screen.

**Score System** – The score text will be updated when a player destroys an alien which will feed in an int value and add it to the score total. The score value will be saved if it’s higher than the previous high score then displays in the high score text from then on until it is beaten.

Class Diagram



Testing Strategies

Functionality Testing – is done to confirm whether the end product works to the specifications. This form of testing mainly has the testers looking for generic problems within the game or its graphics and user interface e.g. game assets, stability issues, AV issues and game mechanics issues.

Performance Testing - is used to determine the application’s overall performance under real-time scenarios and load. Conducting this type of testing helps to ensure whether the present infrastructure allowing the smooth functioning of the game.

Ad-Hoc Testing - is a less structured way of testing and it is randomly done on any section of the gaming application. Specifically, there are two distinct types of ad hoc testing. This kind of testing works on the technique called “error guessing” and requires no documentation or process or planning to be followed. Since Ad hoc testing aims at detecting defects or errors through a random approach, with zero documentation, errors won’t be mapped to test cases.

Report and Repair – Throughout these testing procedures all bugs and errors need to be documented in a report. The development team can then sit down and come up with solutions to fix these bugs/errors. Multiple solutions might need to be implemented and tested to see which solution performs better.

UI and Menus

Score and high score displayed at top in text.

Mystery ship in orange floating above other aliens.

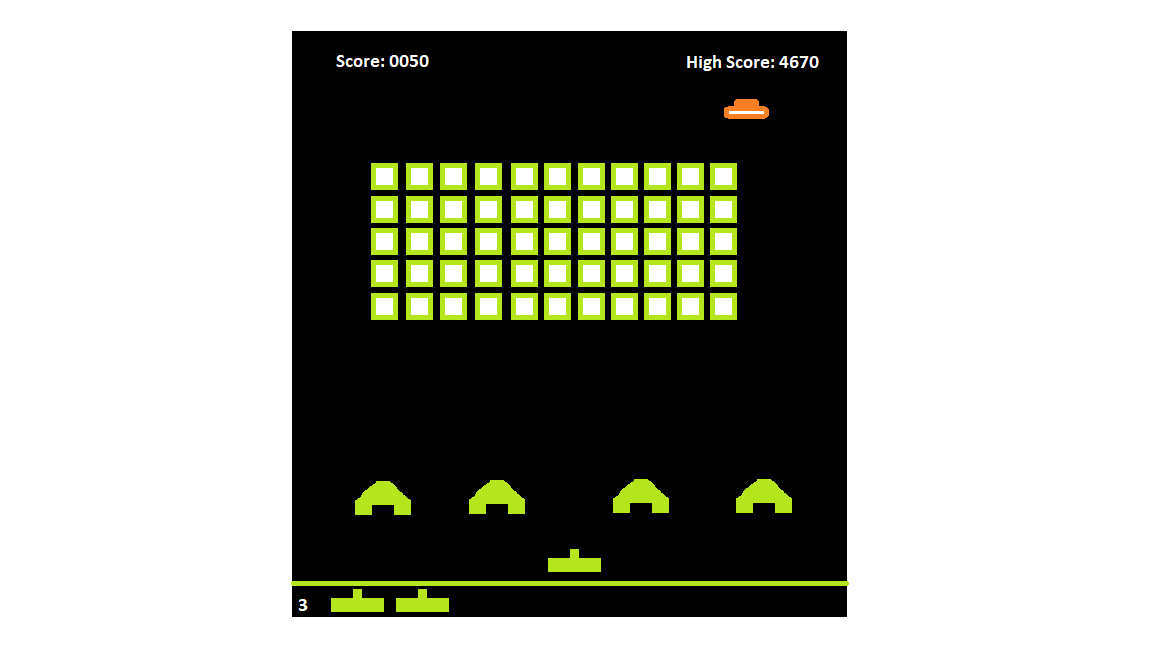
All invaders in green and white slowly moving side to side and down the screen.

Shields placed between player and aliens as green domes for player to hide behind.

Player underneath the shields as rectangle shape object.

Line at bottom for enemy bullets to hit and be destroyed.

Lives displayed in text and image form.



<https://cloud.smartdraw.com/editor.aspx?credID=-39210055&depoId=35485906&flags=128#depoId=35485906&credID=-39210055>

<https://github.com/LegatAbyssWalker/SpaceInvaders>

https://camo.githubusercontent.com/f6e8e17360e68e33efb61bd4b508c1f8185925bf7f5bd11bbb2bf31368d5c472/687474703a2f2f692e696d6775722e636f6d2f3834687863497a2e706e67