# Space Invaders Design document

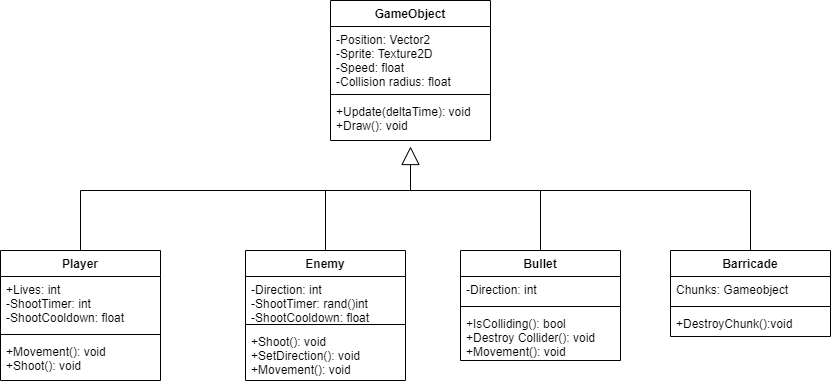
Description of game:

In Space Invaders the player is tasked with destroying waves of alien ships as they descend from the top of the screen, by controlling a laser cannon that is positioned at the bottom of the screen and is only able to move left and right as well as shoot a laser. There are 2 ways for the player to lose, one is by running out of lives, the other is by the aliens reaching the bottom of the screen which will cause the player to instantly lose. The player is given 3 lives when starting the game and will lose one each time they are hit from an alien’s laser that is shot from the ships.

The main challenge of this game is when the alien ships increase in speed as the player get further through the level, making them much harder to hit. The player’s progress is measured by how many enemies they have killed which is displayed as a high score at the top of the screen. After the player has killed all aliens’ ships in the current wave, another wave will spawn lower than the original wave creating a repeatable gameplay loop until the player is defeated.

Also featured in the game are several “barricades” the player can use. The “barricades” are positioned above the player and can act as cover for the player to hide behind to avoid being shot by an alien ship. If the barricade is shot, either by the player or by an alien ship a chunk of it is destroyed and will not be respawned until the next game. This puts pressure on the player as the longer they are alive, the less cover they must hide behind.

Class diagram (UML app.diagrams.net):



Description of programming patterns:

Object pooling- The game will include an object pool to help manage the game objects in the scene. Using an object pool will help with managing the objects by sorting them into two lists, one list being the active game objects and the other being the inactive game objects. By sorting them into these two lists there is no need to waste resources allocating memory every time a new game object needs to be destroyed or spawned. Instead, they are moved between the lists as needed and only the game objects in the active list will have their code executed.

Description of data structures implemented:

Resource manager- A resource manager for the textures will need to be implemented into the game, this will help to reduce the amount of loading that will be done for objects with the same sprite.

Since there will be rows of game objects with the same sprites, having each individual object load the sprite every time they are spawned is a waste of resources. Instead, a resource manager can load the texture once and store it in a hash table. This way any object that needs to load the texture will be referred to the textures address in the hash table instead of having to load the texture from scratch.

Description of algorithms implemented:

Foreach for update/draw

With the implementation of the object pool, looping through the game objects for calling the update and draw functions will be substantially more efficient as the functions will only be called on objects that are in the active list. Every frame the update function will be called on the enemy objects to move them across the screen until the enemies at the border hit the edge of the screen, then the mass of enemies will move lower.

Periodically a function will be called on an enemy to shoot a bullet towards the player. The enemy will be chosen by getting the active ships that are on the bottom row and executing a shoot function at their position.

Collision detection

Since the only objects that will be colliding will be bullets, the only collision checks that will need to be done are from the bullets to the other objects. To optimize the code a spatial hash can be implemented to reduce the amount of collision checks are done per frame. Depending on the type of object that the bullet collides with there will be a different reaction.

For example, if the bullet collides with an enemy and was fired from the player, then the enemy that the bullet collided with will be moved to the inactive list in the object pool and will not have their update or draw function called until a new wave is spawned in which is when they will be moved back to the active list in the object pool.

If a bullet that was fired from an enemy collides with the player, then the players life count will go down and the player will be respawned to continue gameplay. If there are no lives available then the game will end and the players score will be displayed on screen.

If a bullet fired from either the player or an enemy collides with a barricade the section that it collided with will be moved from the active list in the object pool to the inactive pool. From there it will no longer be drawn, checked for collisions allowing for it to be shot through. This puts pressure on the player as the longer the game goes on the less area the player has to retreat from getting fired upon.

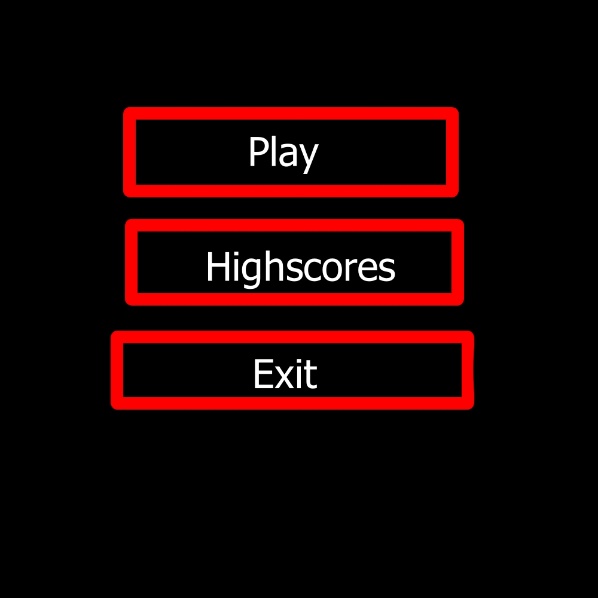
Description of applicable testing strategies:

Methods to test various programs

One way to test the optimization of the various data structures and algorithms is by creating the objects through a for loop for the number of objects needed, instead of implementing an object pool and texture manager. Then seeing how the program handles the frequent loading and unloading as new objects are created and destroyed as the game is played. There will be a significant increase in memory usage when the game first starts and every time a new wave is having to be loaded in. This will demonstrate the need for an object pool and a texture manager.

Mock-ups of user interface and menus:

When the program is first launched the player will be presented with a screen containing 3 buttons.



Play:

Pressing the play button will direct them to the game and they will begin play.



High scores:

As a stretch goal the program will contain a high scores menu to invite an element of competition. The list of scores will be contained on a JSON file that would overwrite a score if the score achieved is greater than one on the list.

