Invadem Report

Project Description

Invadem is a version of the classic 'Space Invaders' first released in 1978 by Taito. It shares many similarities with the original game, e.g. both have invaders traveling down the page shooting projectiles and trying to eventually reach the barriers.

Iteration 1

To fulfill the milestone requirements I implemented various classes and interfaces. These include a *Tank*, *Projectile*, *Barrier*, and *BarrierPart* classes as well as a *Move* interface. All the classes were used to store the necessary information about each game object mentioned in the specification.

In particular, I separated the *BarrierParts* and *Barrier* classes, as I thought it would allow for easier implementation and a modular design. - the BarrierPart class represented the small sections the *Barrier* was made of, whilst the *Barrier* class stored multiple *BarrierParts*.

In the App class, all the game objects were stored in ArrayLIsts to ensure they could be easily accessed when required and that

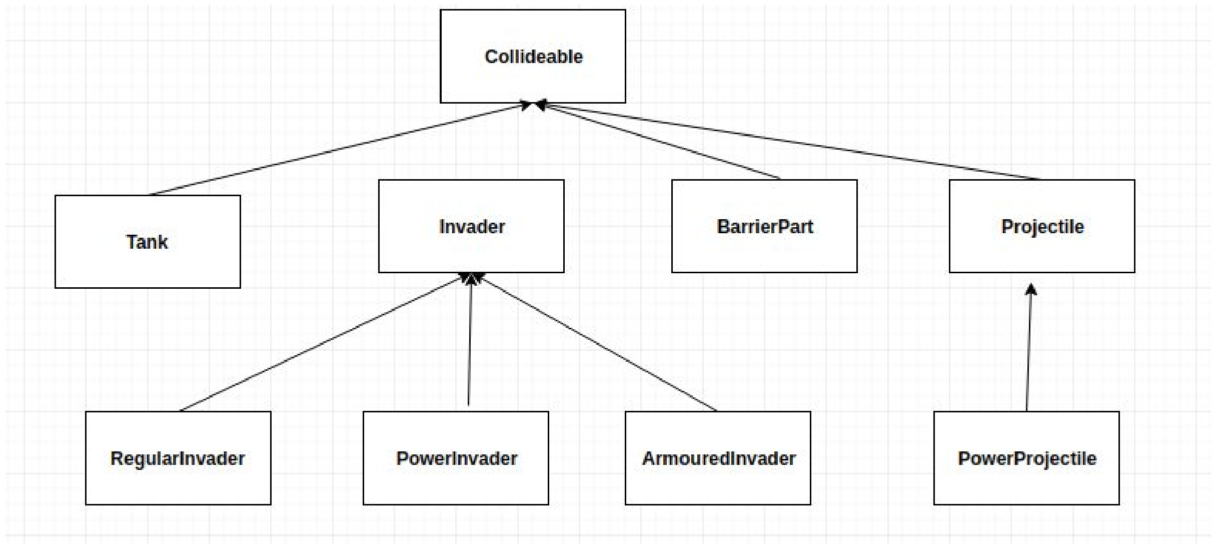
Iteration 2

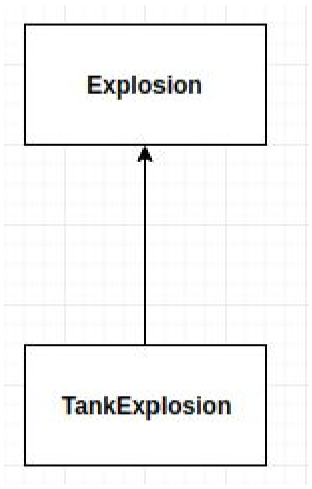
Following the milestone submission, I ended up altering my code design after encountering the problem of incorporating collision detections. Initially, I had appropriately modified the collision detection method given by the specification to determine if 2 objects had collided or not. However, after coding the tank's projectiles and invaders collision, I came to realise that a lot of code had to be written to implement the collision detections between 2 objects. Hence, to avoid the repetition of large amounts of code, I modified my class structure buy adding a new *Collideable* class which was used as a super type to the *Tank, Projectile, Invader, BarrierPart* classes. Through the use of this *Collideable* class I was also able to minimize the repetition of instance variables, methods, and the constructor, as all the aforementioned classes had similar properties. For example, all the classes had a list of instance variables; list of *sprites*, *x* and *y* coordinates, *width*, *height*, *velocity* vector, and *health*. Further, they also had the corresponding getter methods and *isDead()* and *hit()* methods that were used after a collision was detected. Lastly, a static method, *collides()* was also added to check if 2 *Collideable* objects collided. Hence, the introduction of the *Collideable* class significantly reduced the amount of code I had and provided a more modular design, allowing for easier addition of future 'collideable' game objects.

Furthermore, a lot of the my code previous to the milestone submission was reused or moved to another class (eg. *Collideable* class). Further, *nextLevelReset()* and *newGameReset()* were added to the appropriate clear lists of game objects and re-initialised objects to their initial positions. Also, the *Move* interface was removed as it was replaced by the *Collideable* class which allowed for greater functionality.

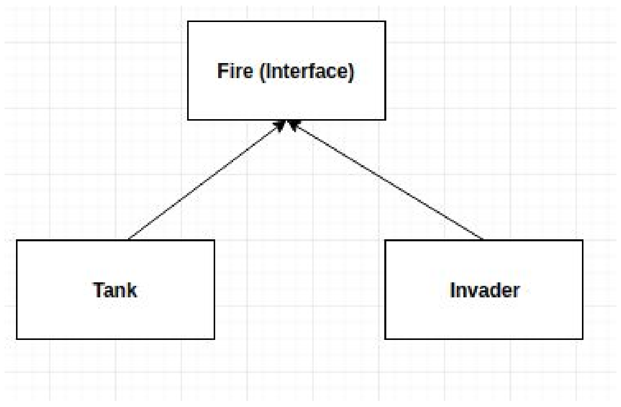
Final Implementation

Class Hierarchy





Interface/Class Hierarchy



As mentioned in the previous section, the *Collideable* class was used as a supertype to the *Tank, Invader, BarrierPart* and *Projectile* classes. This was done to avoid the repetition of code and to add a degree of flexibility to the code design.

Further, an abstract *Invader* class was utilised to act as a super type to the *RegularInvader, PowerInvader,* and *Armouredinvader*. This hierarchical structure seemed natural as the different types of invaders shared many properties; the only ones that differed were their healths, type of projectile fired and sprite.

Similarity, the *Projectile* class was used as a super type for the *PowerProjectile* class, as they share similar characteristics; except for the amount of damage they do and its sprite.

The *BarrierPart* class was used to store information about each section of the Barrier and the *Barrier* class stored a list of *BarrierParts.* Additionally, the *BarrierPart* was given a velocity vector, but this was set to [0,0] during the construction of the class. Whilst, this may seem like an unnecessary piece of code, it allows for additional features to be added to the game in the future, for example, the barriers could move in a specific pattern and at a certain speed, to add a greater level of difficulty.

In addition, the *Fire* interface was implemented by the *Tank* and *Invader* class as they are the game objects which fire projectiles. In this interface there is a method, *fire()* which must be implemented by its sub-types.

Lastly, I significantly changed how my *App* class was structured after the second iteration. Before, my *draw()* method was quite large as it involved all the logic in when/how to draw the game objects at each frame. But after I segmented each section and created methods for each process/piece of logic; which drastically cleaned up my *App* class.

Reflection

<u>Some potential improvements in the code design include:</u>

- Implementing the *Fire* interface in a more meaningful way, instead of just using it for 2 methods
- Maybe utilising an abstract *Projectile* class instead of a regular class - similar to the use of the abstract class for *Invaders*
- Possibility, using more streams to the condense the code and improve its readability

Overall, most of the changes I made throughout the process of building the game seemed to improve either code readability, design, and modularity. Hence, it shouldn't take too much effort to incorporate additional features to the game to create a more engaging experience for the gamer.

Extension

As an extension to the project specification, I choose to integrate explosions into my game. Explosion occur when:

- A projectile hits an invader and it dies
- A projectile hits the tank and it dies

These were added to enhance the game's visual appeal and to make it more enjoyable.