SOFT2201 - Assignment 1

1. Use Case and Use Case Diagram

1.1. Use Case description

Use Case: Shoot an alien.
Primary Actor: Player.
Stakeholders and Interests: Player, Developers.
Preconditions:

- The game is in progress.
- A wave is in progress.
- No bullet is active.

Success Guarantee (Postconditions):

- The alien is destroyed.
- The bullet is destroyed.
- Score is increased.

Main success scenario:

1. The player shoots a bullet.
2. The game takes input from the player and simulates the bullet.
3. The bullet hits an alien.
4. The alien is destroyed.
5. Increase player's score.
6. Bullet is destroyed.
7. The game continues with the player's next action.

Extensions:

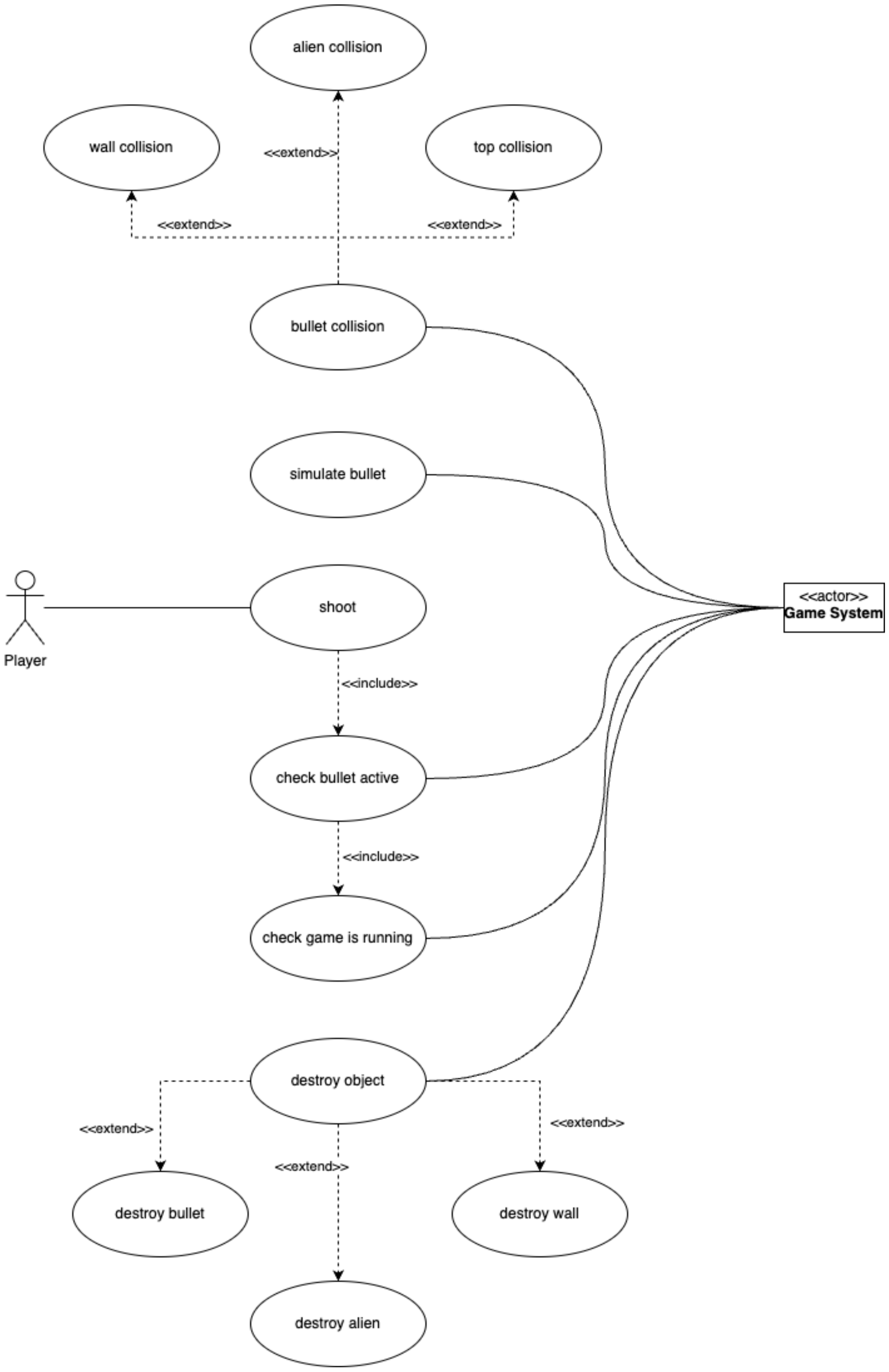
3.a. The bullet hits a wall

1. Destroy the wall.
2. Continue at step 6.

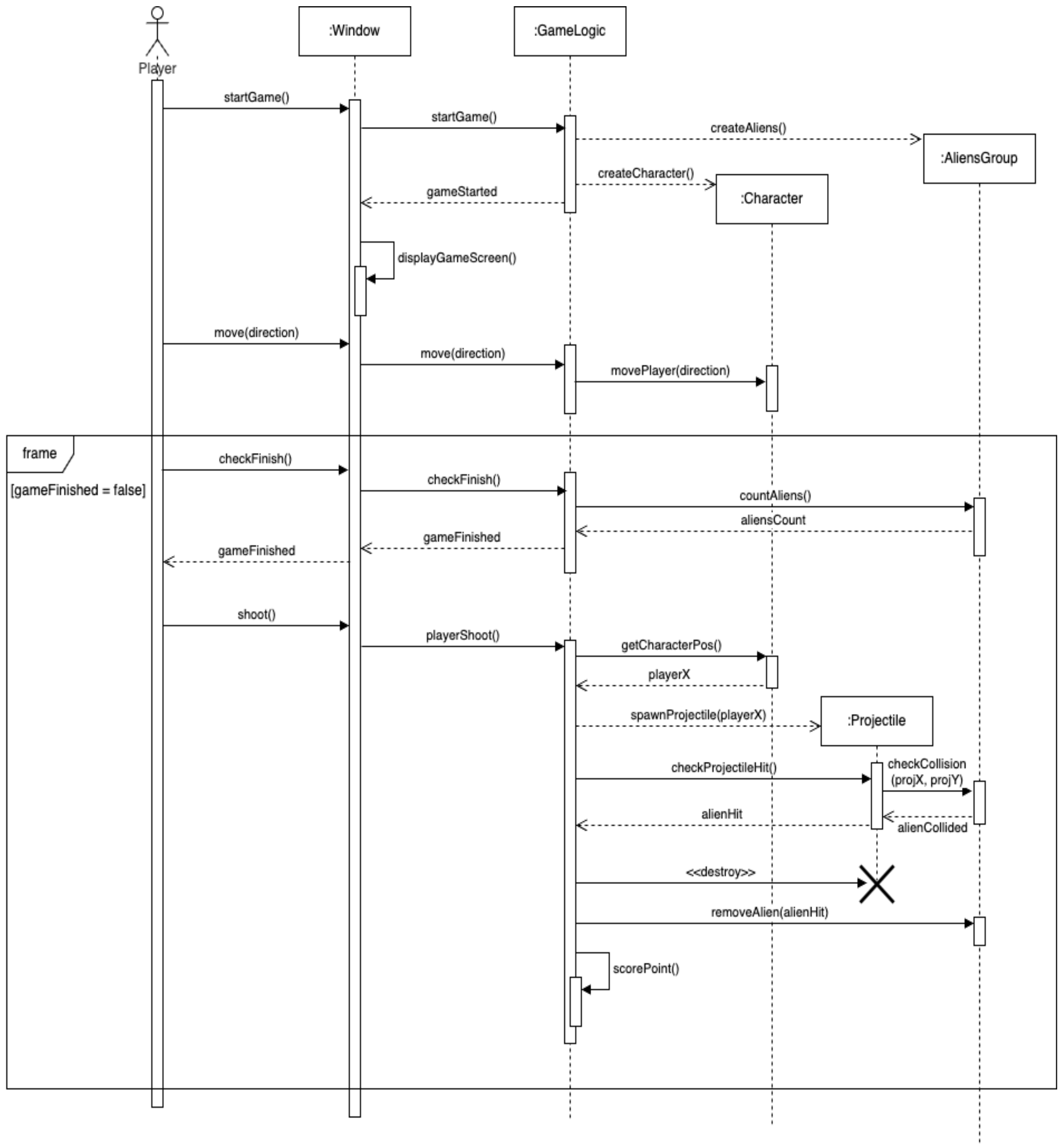
3.b. The bullet hits the top of the screen.

1. Continue at step 6.

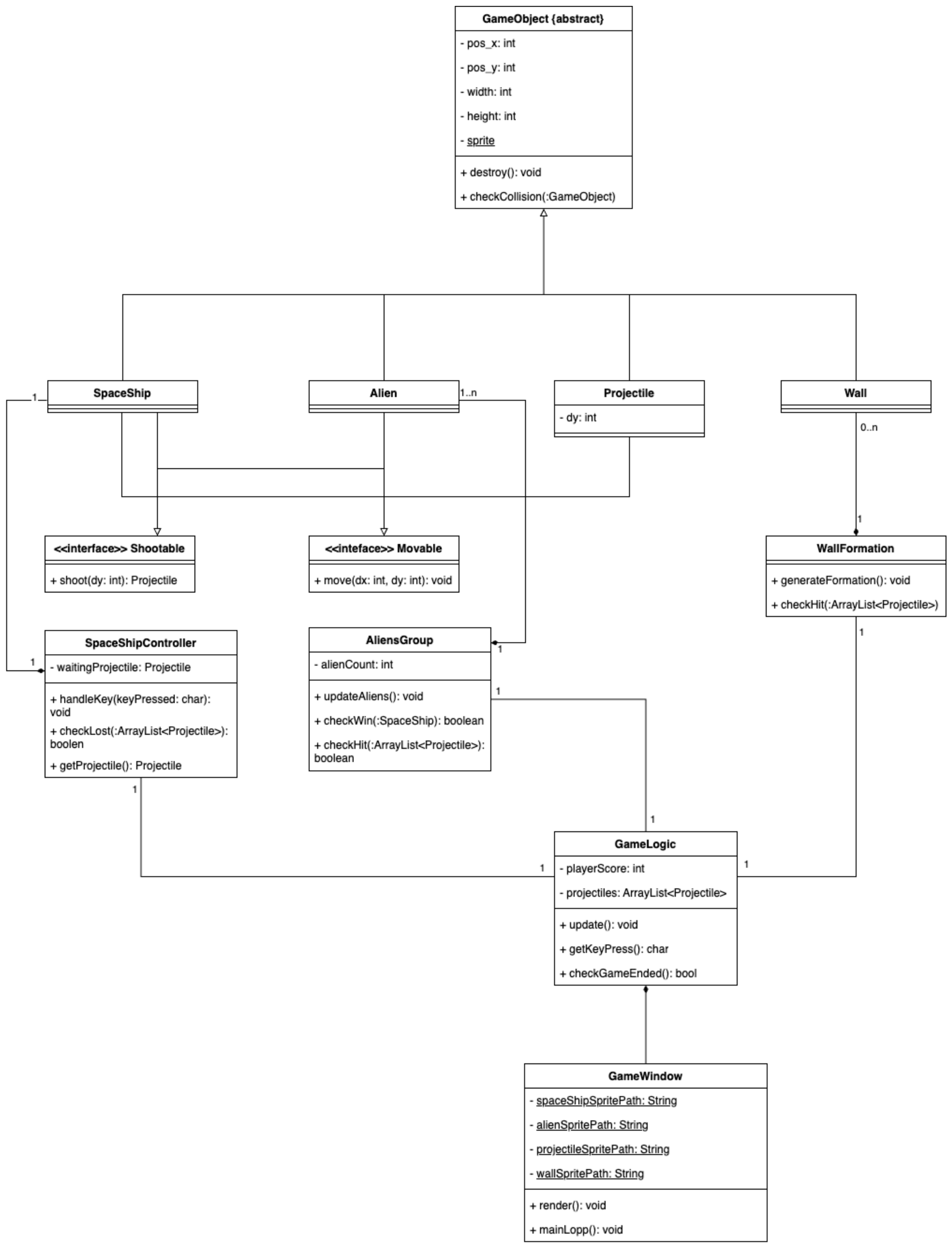
1.2. Use Case diagram


alien collision
wall collision
<<extend>>
top collision
<<extend>>
<<extend>>
bullet collision
simulate bullet
Player
shoot
<<actor>>
Game System
<<include>>
check bullet active
<<include>>
check game is running
destroy object
<<extend>>
<<extend>>
<<extend>>
destroy bullet
destroy wall
destroy alien

2. Sequence Diagram



3. Class Diagram


GameObject {abstract}
- pos_x: int
- pos_y: int
- width: int
- height: int
- sprite
+ destroy(): void
+ checkCollision(:GameObject)
SpaceShip
Alien
Projectile
- dy: int
Wall
1
1..n
0..n
<<interface>> Shootable
+ shoot(dy: int): Projectile
<<inteface>> Movable
+ move(dx: int, dy: int): void
WallFormation
+ generateFormation(): void
+ checkHit(:ArrayList<Projectile>)
SpaceShipController
- waitingProjectile: Projectile
+ handleKey(keyPressed: char): void
+ checkLost(:ArrayList<Projectile>): boolen
+ getProjectile(): Projectile
AliensGroup
- alienCount: int
+ updateAliens(): void
+ checkWin(:SpaceShip): boolean
+ checkHit(:ArrayList<Projectile>): boolean
GameLogic
- playerScore: int
- projectiles: ArrayList<Projectile>
+ update(): void
+ getKeyPress(): char
+ checkGameEnded(): bool
GameWindow
- spaceShipSpritePath: String
- alienSpritePath: String
- projectileSpritePath: String
- wallSpritePath: String
+ render(): void
+ mainLopp(): void

Rationale:

- Abstraction:
    - According to the specification, I chose to represent the Space Ship, Aliens, Projectiles and Walls as objects in the application since they are the most basic entities in the game. These objects model the game entity on a 2 dimensional plane, so it makes

  sense to record their respective positions and bounding boxes, and the basic functionalities include collision checking between objects and objects' destruction.
  - Besides, I chose to create a class to control all of the aliens at once, a class to control all the walls at once and a class to handle player's control over the Spaceship.
  - The game's internal logic is represented in the GameLogic class, this is where the game updates and logic happens.
  - Finally, the game's window (where the player themself interact with the game) is represented in its respective class.
- Encapsulation:
  - All of the classes in my design have their attributes hidden from the outside and only provide the interfaces to interact with those attributes.
  - My design separated different tasks that the application required into different classes to provide modularity.
- Inheritance:
  - I found the game objects (spaceship, alien, projectile, wall) have a lot of similarities, so a superclass can be created, this class is called GameObject.
  - There is a similarity shared among the spaceship, aliens and projectile that is they can move, but this is not true for a wall, so I created an interface for this functionality.
  - Furthermore, the spaceship and aliens can shoot projectiles, so I created an interface for it.
- Polymorphism:
  - The shoot functionality is shared between spaceship and alien, but the implementation should be different and specified to the respective class.