Space Invaders : Sprint 4 Assignment

Group 22
TI2206 Software Engineering Methods

Ege de Bruin
Bryan van Wijk
Dorian de Koning
Jochem Lugtenburg

October 16, 2015

Supervisor: Dr. A. Bacchelli
TA: Danny Plenge

Delft University of Technology
Faculty of EEMCS

1 Exercise 1 - Your wish is my command, Reloaded

1.1 Local co-op Multiplayer Functional Requirements

A list of functional requirements considered for the implementation of local co-op multiplayer using the MoSCoW method described in the previous section.

1.1.1 Must Haves

The local co-op multiplayer must meet the following requirements:

- A multiplayer game shall contain two spaceships.
- The game shall be able to load a multiplayer game.
- The first spaceship shall be controlled using predefined keys on the keyboard.
- The second spaceship shall be controlled using predefined keys on the keyboard.

1.1.2 Should Haves

The local co-op multiplayer should meet the following requirements:

- The User Interface shall have a menu to select between single or multiplayer games.
- The User Interface shall display two different score elements and life elements for different players.
- Each spaceship shall have an equal initial movement speed.
- Powerups work for individual spaceships.

1.1.3 Could Haves

The local co-op multiplayer could meet the following requirements:

- The game shall end after one of both players runs out of lives.
- The game shall display the winner after the game ends.
- The player wins if he/she has the highest score.
- The game shall spawn a powerup with the ability to freeze the other player for a predefined amount of time.

1.1.4 Would/Won't Haves

The local co-op multiplayer won't meet the following requirements:

- Spaceships shall collide.

1.2 Class responsibilities and collaborations

The following new classes were created for the implementation of the new local multiplayer feature. In the table the responsibilities and collaborations are presented for every class.

Class	Responsibility	Collaborates with	Super	Sub
MultiPlayerGame	Players		Game	
GameUIController	drawing the UI elements	MultiPlayerGame	GameUIController	
MultiPlayerGameUIController	Key control	MultiPlayerGame	GameUIController	
SinglePlayerGameUIController	Key control	Game	GameUIController	
MenuUIController	Conrol menu	MultiPlayerGameUI-Controller		
UiElementMultiSpaceShips	draw multiple spaceShips	MultiPlayerGameUI-Controller		
MultiGameScore	score display multiple players	MultiPlayerGame	Score	
MultiGameLives	lives display multiple players	MultiPlayerGame	Lives	
MultiSpaceShipsController	Control multiple spaceships	SpaceShip	SpaceShipController	
MultiPlayerPowerUpController	Control powerups for multiple players	MultiPlayerGame	PowerUpController	

Furthermore we have to replace the check if the alien is at the same height as a spaceship from AlienController to the SpaceshipController.
And replace the score add call to the multiplayercollisions.

1.3 Multiplayer UML

The UML is as following:

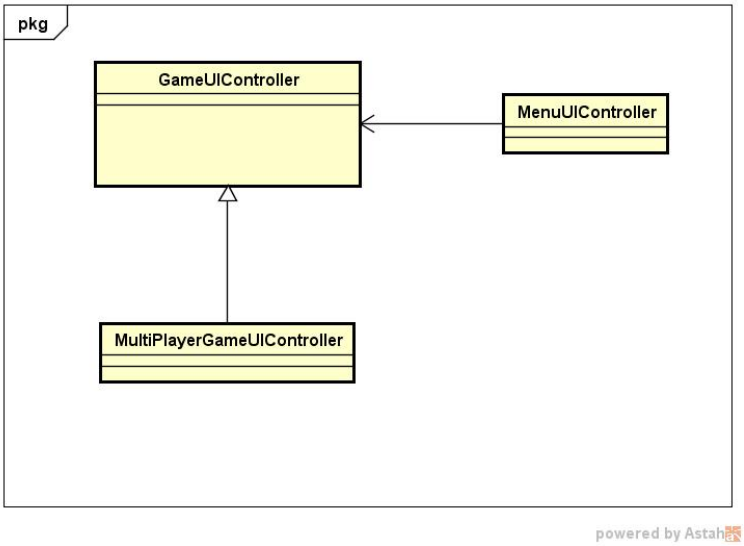

Figure 1: MultiGame UI UML

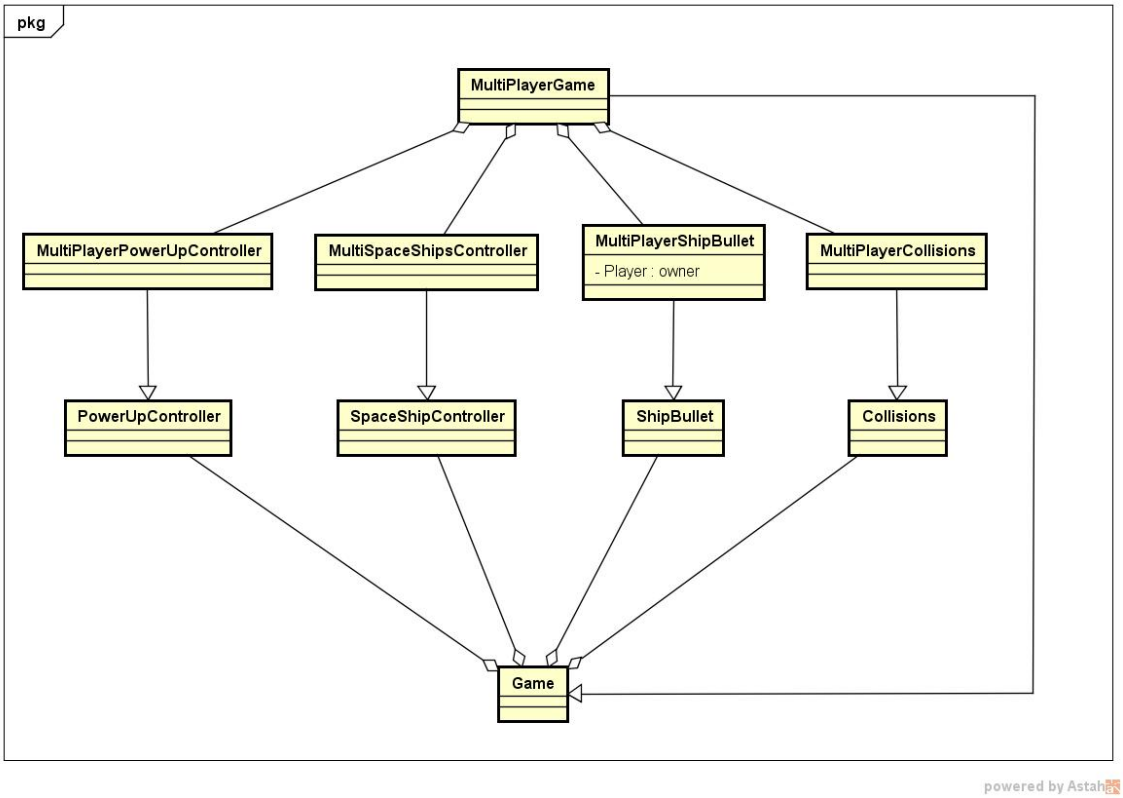

Figure 2: MultiGame UML

2 Exercise 2 - Software Metrics

The inCode tool is used to find design flaws, the resulting analysis file can be found in the sprint4 map in the deliverables map in the git resository. The first design flaw was encountered in AlienController, with a Cumulative Severity of 5. AlienController contained Message Chains in the move() method leading to extensive calls to external accessors. It is caused by calling a method, which returns an object, calling a method on that object and so on. This method contains a chain of method calls to obtain the spaceship. This chain can be reduced by delegating this 'chain' to the Game class, so AlienController needs less knowledge about how to obtain the spaceship from the player. This was done by creating a new method, getPlayerSpaceship() which returns the Spaceship obtained from the player.

The second and third design flaw were encountered in Collisions and SpaceShipController. These methods also contain Message Chaining, caused by the same message chain as in AlienController. This could be solved by calling the method delegated to the Game class instead of the message chain.

Because the first three design flaws were relatively easy to fix, other design flaws were also fixed. UIElementSpaceShip was also affected by the same chain of messages, this was resolved by calling the method created in the Game class. SpeedPowerUpUnit, ShootPowerUpUnit and LifePowerUpUnit contained feature envy in the equals methods. This was resolved by moving the duplicate equals code to the superclass and extending it correctly. UIElementBullet contained a feature envy in draw(). This was resolved by moving duplicate code to the superclass UIElement and splitting the calculation of x and y positions into two separate methods.

Because of the merge of the features implemented in Exercise 1, all inCode errors have now been resolved.