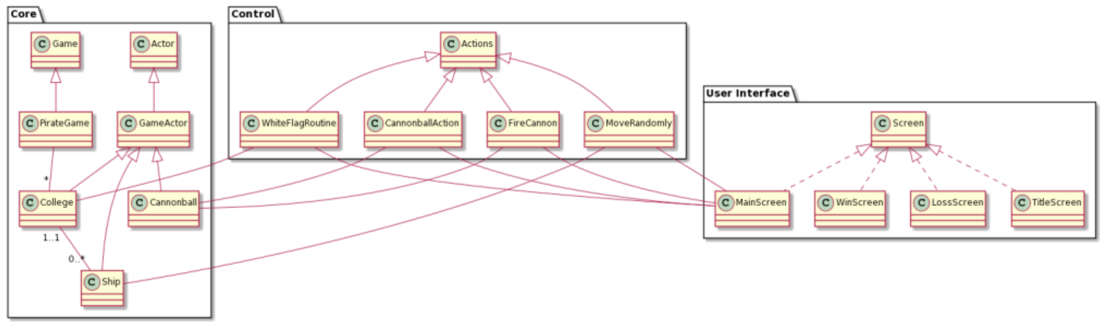
Architecture

Overview

This document presents the architecture, which is based on an underlying Unified Modelling Language (UML) model developed using `plantuml.com`. Our design decisions were carefully thought out and refined by comparing other existing pirate games. We started with an abstract architecture using a package diagram, then went into more detail about the classes and relationships between them.

- Single-player realtime game set in open waters and the player is a privateer with allegiance to a college.
- There are 4 rival colleges around the lake, each populated by a number of pirates. The player sails around on a ship, whilst encountering ships from other colleges.
- The player begins with a ship which contains a cannon.
- As a privateer, the aim is to expand their territory and take over all the rival colleges. Over time, the player can accumulate experience points. As points are accumulated, they can gain skill upgrades such as faster ship movement.
- The player may battle rival colleges, by firing cannonballs until the college is neutralised and the player gains experience. Upon college destruction, the player gains gold instead. (This is explained more in the scenarios section).
- A rival college will fire cannonballs at the player, and if the player ship is hit, it will result in loss of health. Once the player's health hits 0, the game ends, and the player loses.
- Alternatively, if the player defeats all rival colleges, the game ends and the player wins.
- Enemy ships can shoot at the player and if the player ship is hit, it will result in loss of health. Once the player's health hits 0, the game ends, and the player loses.

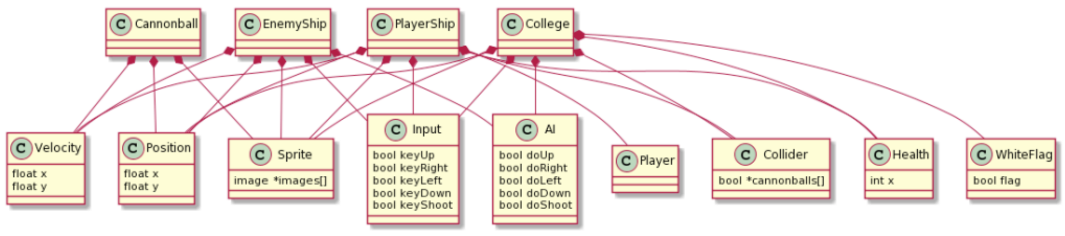
The architecture of this game will be based on three layers: **Core**, **Control** and **User Interface**. The package diagram below represents a low level view of how the layers interact.



The three game objects college, ship and cannonball are each initialised with at least one class from **Control**. The actions in the **Control** layer control what is displayed on the main screen.

Game logic

The game entities are the **Ship**, **College**, each of which can be associated with the Player or Enemy. Below is an entity-component diagram which shows what the data structures each object has.



- PlayerShip (Sprite, Input, Player, Position, Velocity, Health, Collider)
- EnemyShip (Sprite, Input, AI, Position, Velocity)
- College (Sprite, Input, AI, Position, Health, Collider, WhiteFlag)
- Cannonball (Sprite, Position, Velocity)

Systems

- Render (Sprite, Position) Draws sprites at a position
- Movement (Position, Velocity) Modifies direction of ship based on key pressed and updates x and y coordinates by the velocity
- PlayerControl (Input, Player) Sets the player ship input according to keyboard controls: WASD to control movement and spacebar to shoot
- Defeat (Health, WhiteFlag) Sets college white flag when it is defeated (health is at 0)
- BotControl (Input, AI) Set the enemy entity's input according to an AI agent
- PlayerVelocity (Velocity, Experience) Speed of player ship is determined by some function of the experience value
- Damage (Collider, Health) If the player ship or a college is in contact with a collider (cannonball), then modify its health value.
- Weather : Changes the players screen to display the weather.

Concrete architecture

All the classes either implement or extend an existing class from libGDX namely Screen, Actions, Game and Actor. Where relevant, a static perspective of the class and its functions are shown.

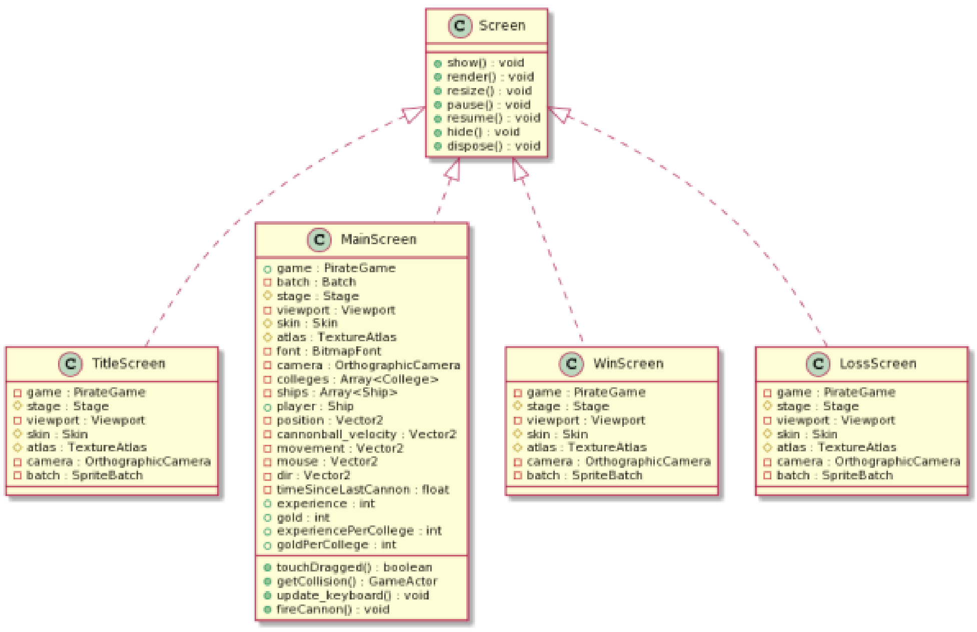
User interface

There are four screens in total:

- The title screen which is displayed upon game run,
- The main screen which is where the pirate game is run,
- The win screen and
- The lose screen which are displayed depending on the game outcome at the end.

The four screens implement the class Screen from libGDX, which already contains the methods we need.

The following represents the class diagram for the graphical user interface.



Title screen

The title screen view is displayed at game start, and when the game is escaped. It gives access to the following actions:

- Play (which sets screen to **MainScreen**)
- Quit (which exits the game)
- Difficulty (Choose how hard the game will be)

Underneath the menu, there are gameplay instructions and keyboard controls.

Main screen

The main screen is where the game takes place. Stats such as health, experience and gold are viewable at a corner of the screen.

- **Esc** exits the game and displays **TitleScreen**

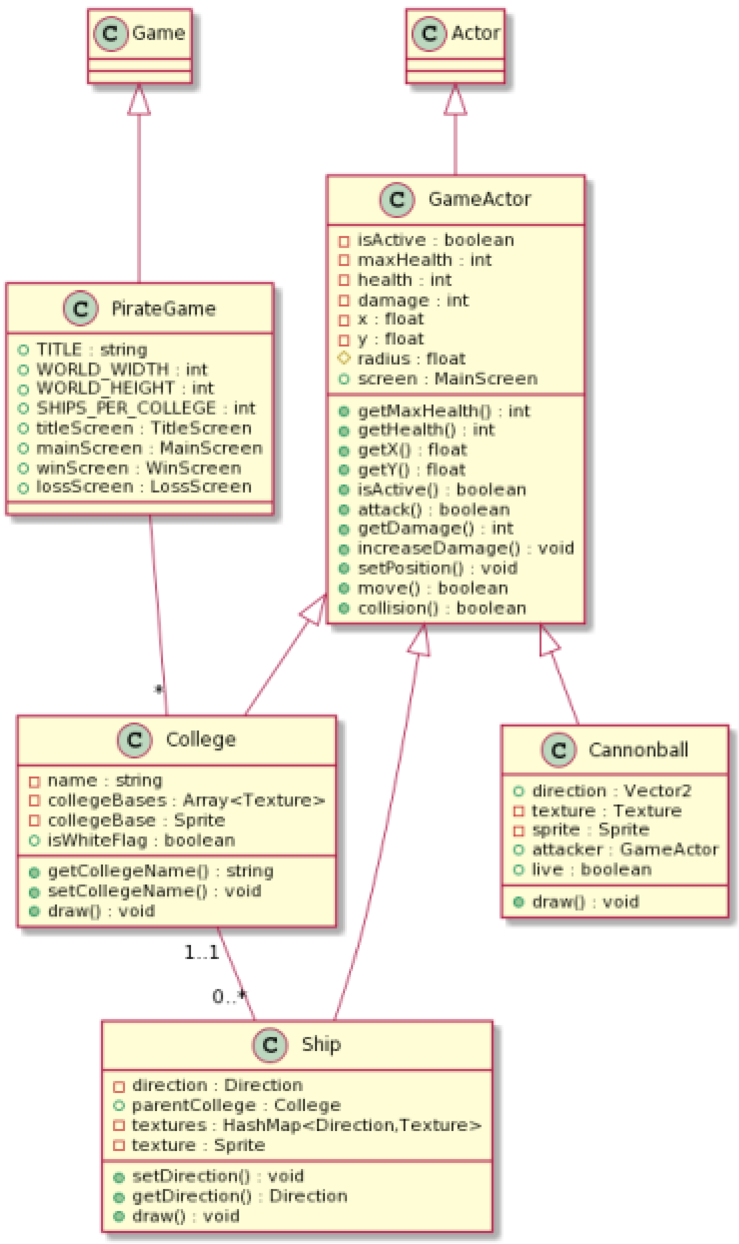
On this screen, keyboard inputs are recorded and trigger ship movement or a cannonball to fire.

Outcome screen

When the game ends, either **WinScreen** or **LossScreen** is displayed with a message. It gives access to the following actions:

- Play Again (which sets screen to a new instance of **MainScreen**)
- Quit (which exits the game)

Core



- PirateGame is initialised with four screens (**TitleScreen**, **MainScreen**, **WinScreen**, **LossScreen**) and sets the current screen to **TitleScreen**.
- Each game object College, Ship, and Cannonball inherit from a custom class **GameActor**. Very simply, initialization sets up "per instance" member variables and loads "per class" textures as seen in the diagram. Moreover, each game object will have at least one action (from the **Control** layer).

```
CCollege
      Purpose: Implement the basic College class which defines
functionality common to all colleges.
```

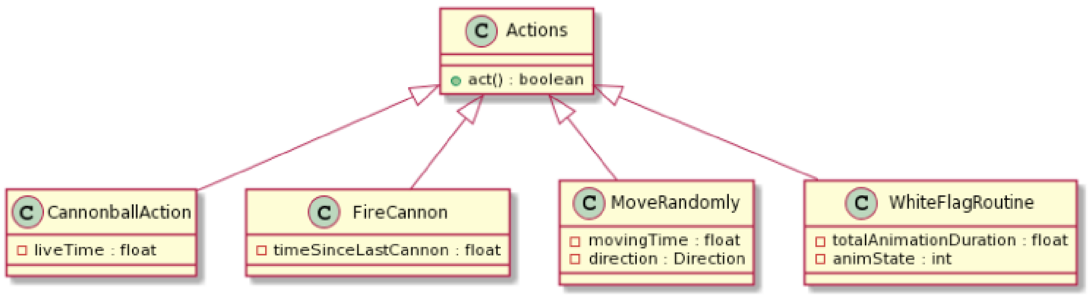
```
Function CCollege::College
      Function: Performs initialization of the college object.
Adds action FireCannon.
      Inputs: MainScreen, String
      Outputs: None
      Returns: None

Function CCollege::attack
       Function: Attacks the college and deals damage, where the
magnitude of the damage is set by the input value. Adds action
`WhiteFlagRoutine` when the health reaches 0.
       Inputs: Integer
       Outputs: None
       Returns: None
```

```
CShip
      Purpose: Implement the basic Ship class which defines
functionality common to all ships.

      Weapons: Single-shot cannon

Function CShip::Ship
      Function: Performs initialization of the ship object. Adds
action MoveRandomly if the ship is not PlayerShip.
      Inputs: MainScreen, College, isPlayer
      Outputs: None
      Returns: None

Function CShip::setDirection
       Function: Sets the direction the ship is facing
       Inputs: Direction
       Outputs: None
       Returns: None

Function CShip::getDirection
       Function: Get the current direction the ship is facing
       Inputs: None
       Outputs: None
       Returns: Direction
```

```
CCannonball
      Purpose: Implement the basic Cannonball class which defines
functionality common to all cannonballs. This class is intended to
be instantiated as a concrete class.
```

```
Function CCannonball::Cannonball
      Function: Performs initialization of the cannonball object.
Adds action CannonballAction.
      Inputs: MainScreen, float, float, Vector2, GameActor
      Outputs: None
      Returns: None
```

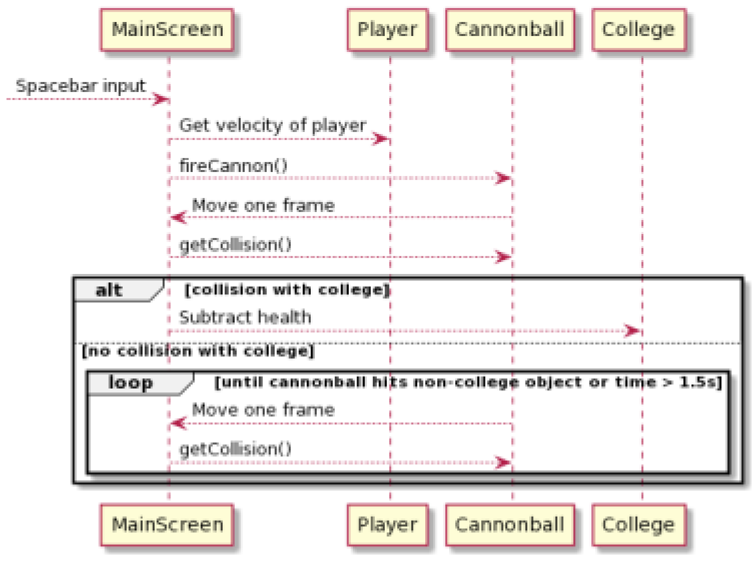
Control



- **CannonballAction** moves the cannonball and checks for a collision on the screen (using the **MainScreen** method **getCollision()** which is shown on the User Interface class diagram.)
- **FireCannon** defines the AI input for college combat - it will fire a cannonball at random intervals in the direction of the player ship.
- **MoveRandomly** defines the AI input for NPC ship movement.
- **WhiteFlagRoutine** is added to a college once it reaches 0 health. Then the player is able to either capture or destroy the college.

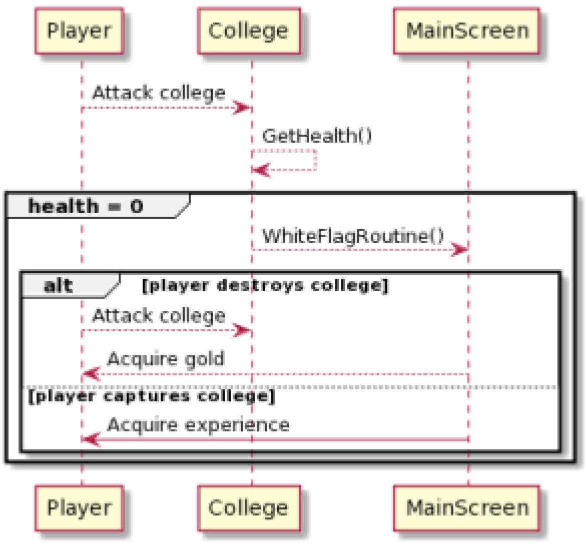
Scenarios

Scenario 1: CollegePlayer combat



- The **MainScreen** class records a spacebar input, which calls the **fireCannon()** method. This method initialises a new cannonball object with **CannonballAction**.
- A cannonball is spawned and is set moving towards its target. If a valid GameActor (player ship, college) is hit, then damage is applied.

Scenario 2: CollegeDefeat



- College health reaches 0 after sufficient cannonballs have been fired at it. A white flag is acquired by the college.
- If the player attacks it within the next 10 seconds, the college is destroyed and the player acquires gold.
- If it is not attacked, then it will be captured (join the player's college) and the player acquires experience.
- Ships from the college join the player’s college

Scenario 3: ShipPlayer Combat

- Ships shoot at player all at the same time for gameplay purposes - was too difficult if they shot random intervals
- Player can shoot at ships to destroy them

Scenario 4: ShipDefeat

- Ship health reaches 0 after sufficient cannonballs have been fired at it.
- If the player attacks it within the next 10 seconds, the ship is destroyed and the player acquires gold.

Architecture Justifications

Requirement	**Justification**
UR_FAIRNESS	Four enemy colleges to destroy - this will not take too long or be too difficult. For consistency between replays, attributes such as ships per college and the number of colleges are constant.
UR_CLEAR_GRAPHICS	Similar to above, we chose four enemy colleges, to prevent over-crowdedness on the map.
UR_EASY_TO_UNDERSTAND	For convenience and easy playability, the gameplay and controls are on the title screen rather than adding an Options menu.
FR_START_BUTTON	There is a **Play** button on the **TitleScreen** to start the game.
UR_CONTROLS	Input controls use the WASD and spacebar keys.
UR_SAILING, UR_COLLEGE_COMBAT	Controls are used for ship movement and firing.
UR_COLLEGE_COMBAT	Colleges will shoot cannonballs at random intervals.

UR_UPGRADES	Ship speed increases as experience is gained.
UR_MUTE_SOUND	At any point, the sound/music can be toggled with keypress **m**.
UR_COLLEGE_CAPTURE	After a college is defeated, they acquire a white flag to indicate it is captured.
UR_COLLEGE_DESTROY	After a college is defeated, attacking it will destroy the college to rubble.
UR_WIN	After all colleges are destroyed, **WinScreen** is displayed.
UR_GOLD	Gold is acquired by destroying enemy colleges.

Note that the view is at a 45 degree angle. Visually, each college and their ships can be identified by the colour of their flag. Due to our choice of angle, and most existing assets using a birds eye view approach, the art is hand-drawn (i.e. ships and colleges).

References

Garlan & Shaw (1994). "An Introduction to Software Architecture"

ISO/IEC/IEEE (2011). "ISO/IEC/IEEE 42010:2011 Systems and software engineering - Architecture description"

Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice, Third Edition. Addison Wesley, 2012