Architecture - Arch1

Dragonite
Team 21

Omar Omar
Rhianna Edwards
Okan Deniz
Craig Smith
Omar Galvao Da Silva
Joel Wallis

Architecture

Tools

The tool used to create the diagrams for the architecture diagrams is the website: https://plantuml.com/. This is because it provides a lot of flexibility in making the diagrams in order to present all the necessary information.

The language used in the website is plain text language which has a relatively simple syntax so it does not take long for training before being able to properly use it in order to create visual diagrams. The website also had useful relevant tutorials on making architectural designs to help learn the syntax. In addition, this website is open source, meaning that no licenses were required.

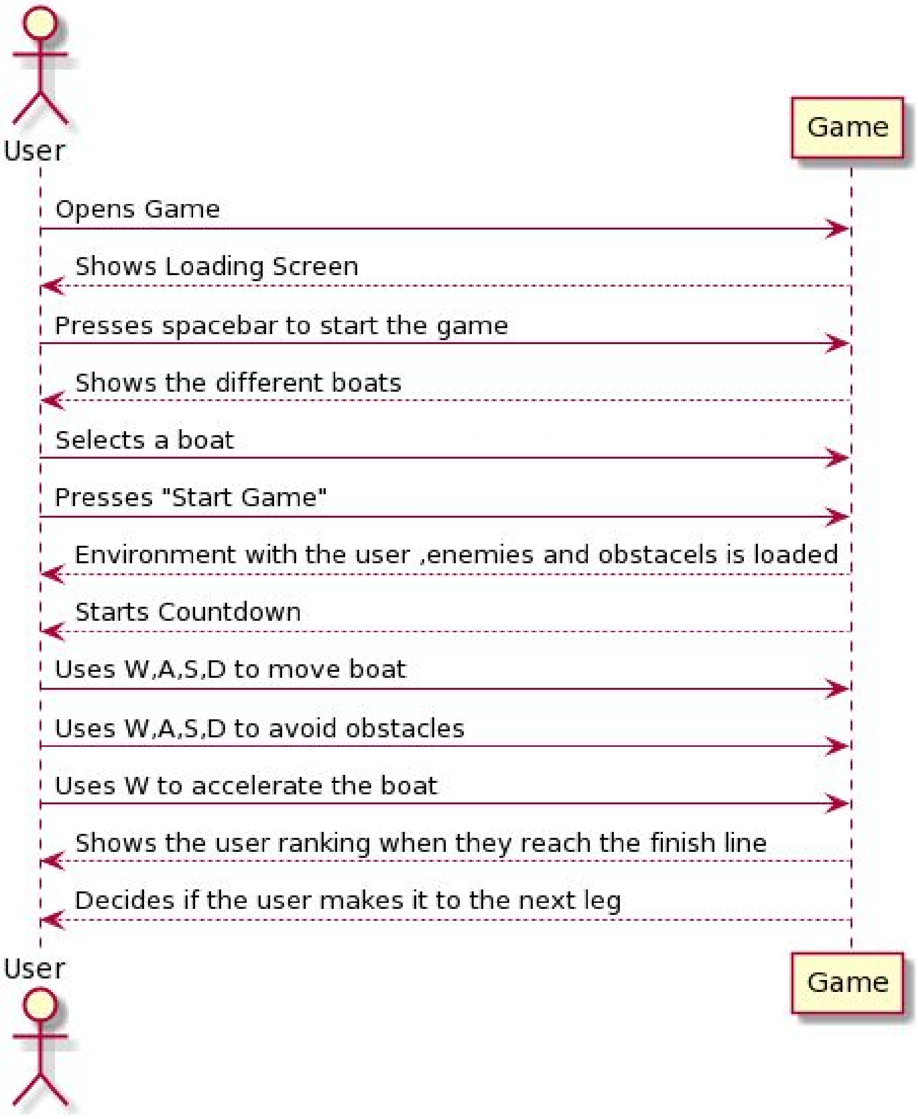
Other tools were available to create the diagrams, however after consideration, we came to the conclusion that this is one of the most user friendly tools and allowed us to easily update our designs when necessary.

As for the flow chart, https://app.diagrams.net/ was used as it provided all the necessary tools in order to make clear flow charts. This was useful for moving shapes around and labeling branches which included decisions. The website is very user friendly and allowed for the diagram to be saved as an editable version so it could easily be altered as well as a PNG format to be used in the documentation.

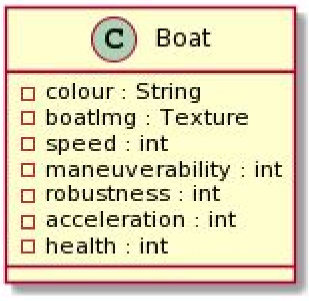
Abstract Model

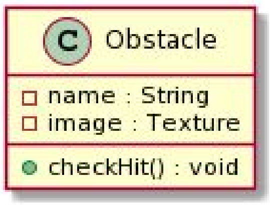
User viewpoint

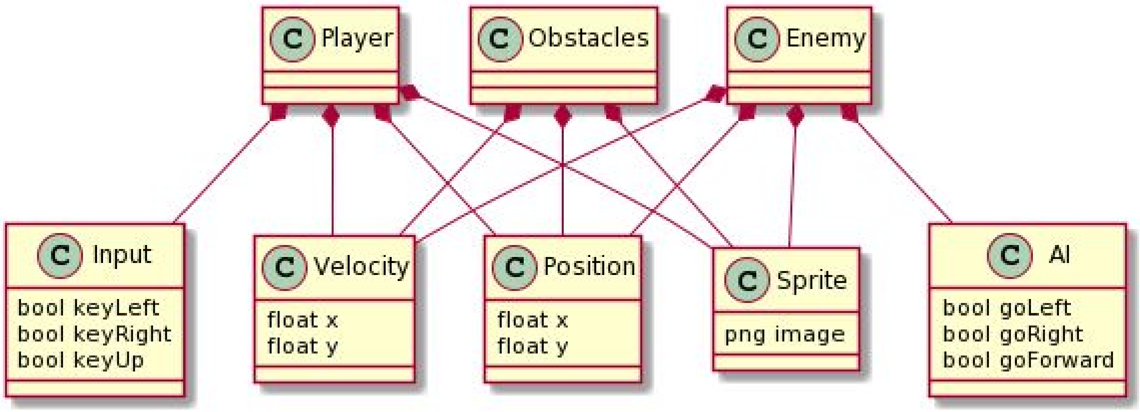
This is a simplified version of how the user would interact with the


User
Game
Opens Game
Shows Loading Screen
Presses spacebar to start the game
Shows the different boats
Selects a boat
Presses "Start Game"
Environment with the user ,enemies and obstacels is loaded
Starts Countdown
Uses W,A,S,D to move boat
Uses W,A,S,D to avoid obstacles
Uses W to accelerate the boat
Shows the user ranking when they reach the finish line
Decides if the user makes it to the next leg
User
Game

Abstract class diagrams

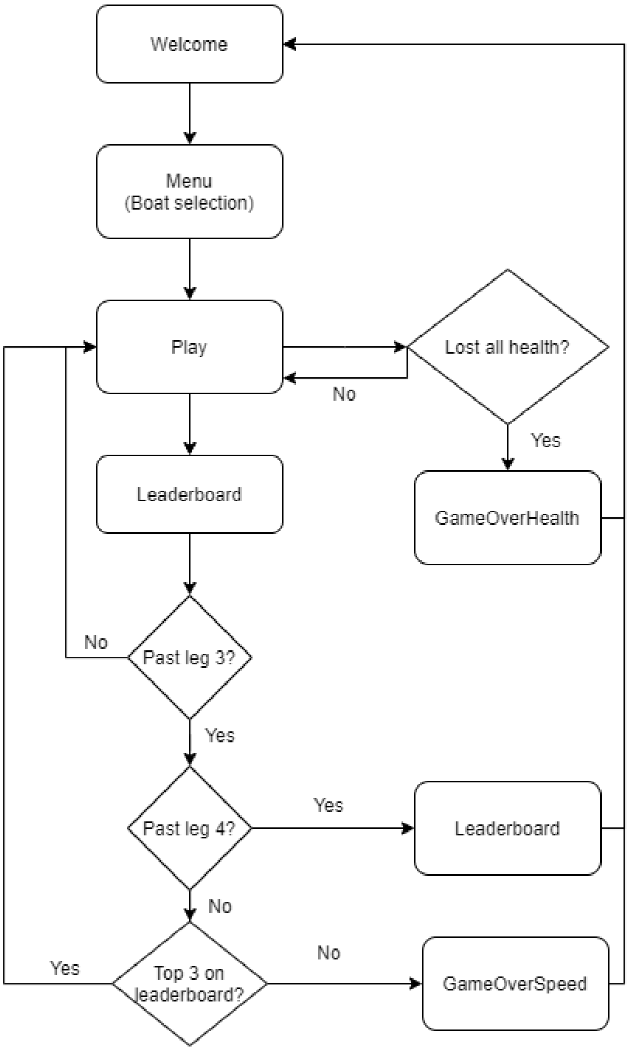






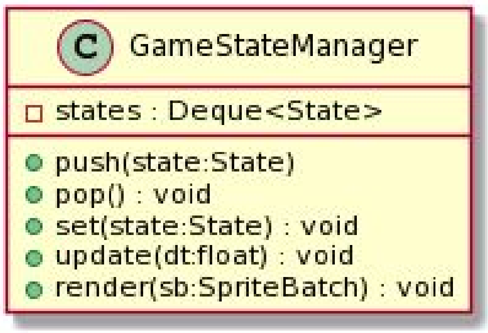
Concrete model

Game Flow

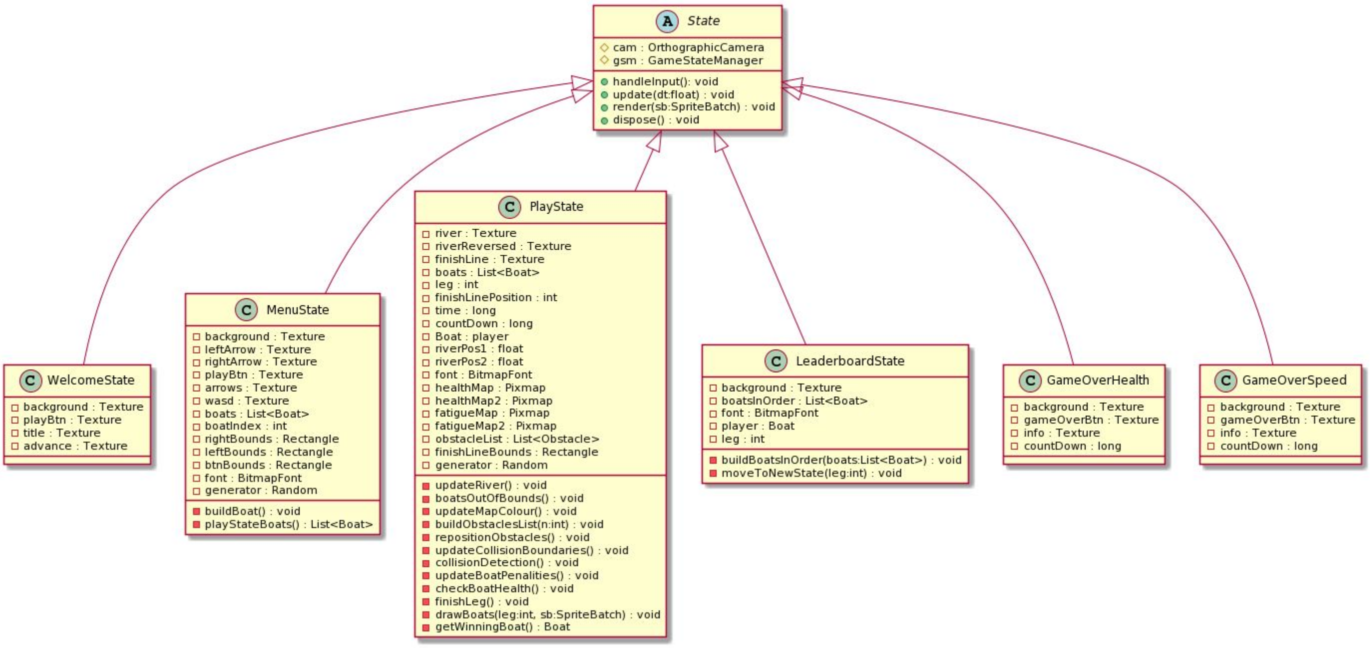


Game state classes

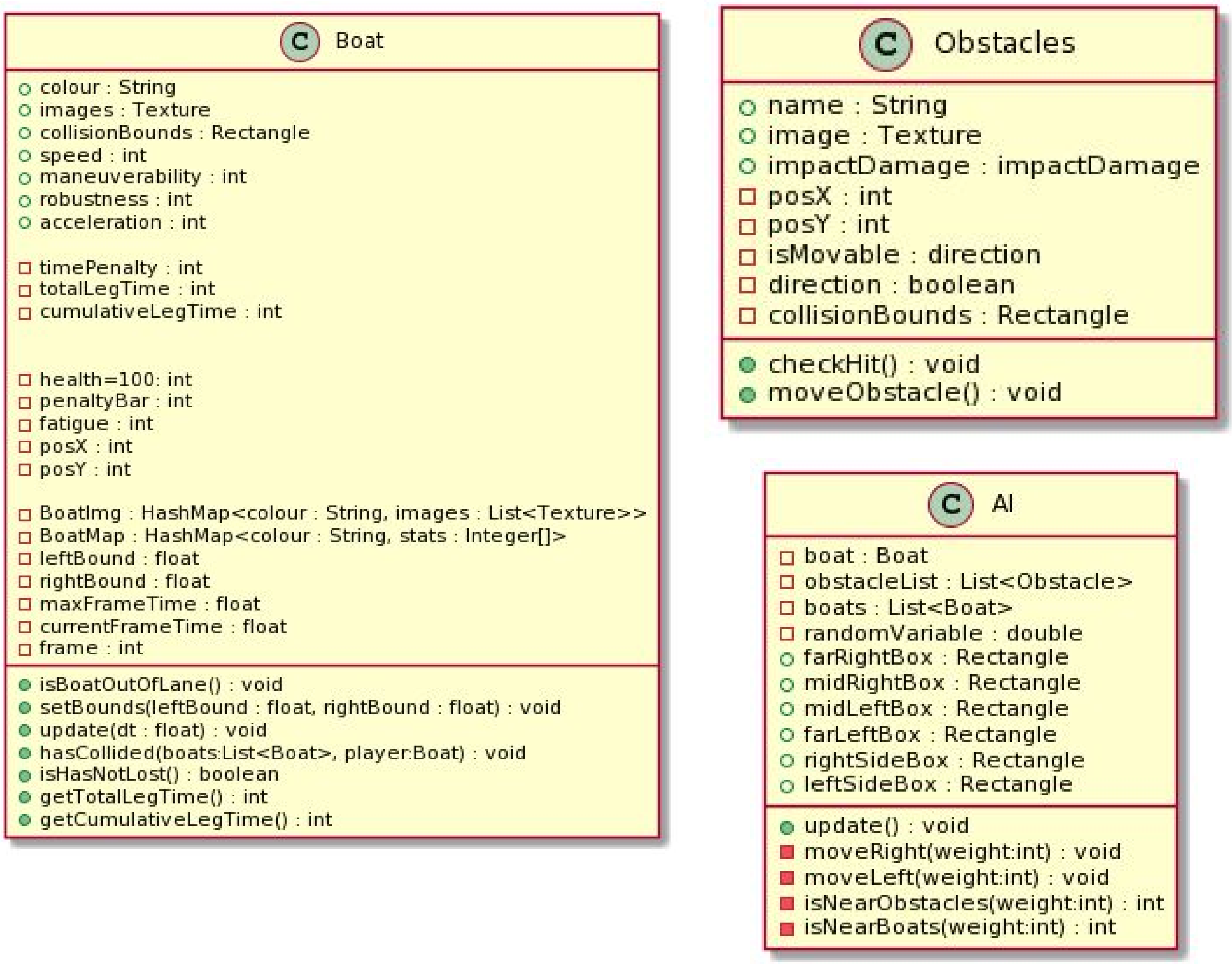
This is used to change between states



GameStateManager
□ states : Deque<State>
● push(state:State)
● pop() : void
● set(state:State) : void
● update(dt:float) : void
● render(sb:SpriteBatch) : void



Class diagrams for boat, obstacles and AI



Systematic justification of abstract and concrete structures

Abstract to Concrete

After experimenting and researching similar games related to this one, more features were added and some things were adjusted for faster and more efficient implementation, such as having all the stats for the boats into a HashMap which then sets it for each attribute. This is so that rather than having a constructor that takes in many parameters for all attributes, it only needs to take in the colour of the boat. More features such as methods and variables were added because more methods were needed in order to fulfil the requirements. The abstract structure forms a base to build on for the more concrete model as it requires more consideration about how it is actually implemented and how it would affect the fulfilment of the requirements.

Game states were added so it allowed for easy transitions for each stage of the game and allowed the state to be reused as some states were visited more than once.

Relating Concrete architecture to requirements

Requirement	Justification
UR01, SR01	The architecture is designed using Java and libGDX in mind, object oriented programming is used as Java supports it and it was useful to write more reusable and efficient code
UR02, SR02	Timers are used in the PlayState and individual times are saved in each boat object in the attributes in totalLegTime and cumulativeLegTime in order to see which boat has the fastest time. A timer is also available on the screen in Playstate so the player can see how long they are taking to finish the race including any time penalties implemented.
UR03	The user interaction is kept simple, as shown in the diagram above because no prior computer science knowledge is assumed for prospecting students and their families.
UR04	The AI class is made so it controls movement of the AIs that are competing against the player. This class allows the object to detect if there is anything around the front of the boat and so navigates the boat away from obstacles and other boats to have the minimal amount of penalty. However a random number generator is used so it is more realistic and there is a chance that an AI boat will collide with other obstacles. As all the other boats other than the player are AI and there are 5 lanes, there would be 4 AIs for the first 3 legs
UR05	The finish line bound in playState can be adjusted so that the time is just right and between 3-7 minutes.
UR06	The Boat class has a hashmap called BoatMap which contains all the stats for each colour of the boat so the constructor can just only have one parameter - the colour and using the colour as the key, all the values of the attributes can be retrieved from the hashmap. This allows for easy creation of new boats and each different colour boat will have a set of different stats for their speed, acceleration, maneuverability and robustness.
UR07	The boatsOutOfBounds method in Boat checks if the boat is out of lane and penalises them.

UR08	The collision detection in the PlayState would remove the obstacles and also update health/robustness of the boat.
UR09	The checkBoatHealth method checks if the health is 0 and if so changes the game state to GameOverHealth state
UR10	In the Boat class diagram, there is an attribute called fatigue and there would be getter and setter methods for it, when the user presses down on a key in PlayState, the fatigue would make the player go slower and decrease in maneuverability.
UR11	In the MenuState the controls are displayed to show the user how to control the boat and which keys to press on the keyboard in the wasd texture.
UR12	The player boat is in the centre of the screen so it will always be in the view of the user - using the OrthographicCamera in the abstract class State. Also, the boat selected by the user in MenuState will make it obvious that it is the player’s boat as there cannot be 2 boats with the same colour so the AI boats will look different to the user boat.
UR13	As shown in the game flow diagram, after each leg the LeaderBoardState will be shown to give the times of all the boats in the leg.
UR15	Animation of the boat is made in the Boat class. The hashmap BoatImg stores the image of each boat in two states of moving - one with the oars forward and one with the oars back. Then the attributes currentFrameTime and maxFrameTime are used to track and change the Textures using the update method.
SR03	WASD / Arrow keys are used in order to control the player’s boat. The key detection will be implemented in the main game loop as it is required every frame.
SR04	The AI class is created to control the movement of the AI boats. In the update method in the PlayState, all the AI boats will move using the update method in the AI class which determines where the AI boat will go.
SR05	The Boat class has all the attributes and functions required in order for the game to be run. The boat class has many attributes storing the values for the different attributes and all the necessary methods for collisions and moving bounds so after the boat moves, collision detection still works.
SR06	In the render method inherited from the abstract State class, all the assets will be displayed using the render method, showing all the assets including: background/river, boats and obstacles when required.
SR07	Six different game states are in the concrete class diagrams, each showing the relevant information to allow for easy interaction with the game.
SR08	The Obstacle class would store all the relevant information shown in the diagram, such as name, image, position, directions and bounds in order to detect collisions and move obstacles if they are movable.
SR09	The AI class manages the movement of the competing boat by checking if the spaces in front of the boat have obstacles or boats(isNearObstacles, isNearBoats method) and then using a random variable(chance of collision), it would decide rather or not to collide into the obstacle. Further into the race, the random variable will be reduced each leg so there is a higher possible chance that the AI will dodge the obstacle.

References

1. https://www.researchgate.net/publication/228962891_Required_and_Optional_Viewpoints_--_What_Is_Included_in_Software_Architecture