Implementation

Team 14 - Taken On By Team 15

Joe Wrieden

Benji Garment

Marcin Mleczko

Kingsley Edore

Abir Rizwanullah

Sal Ahmed

Introduction

The generation of our code firstly stems from the requirements that have been generated and revised with our customers/stakeholders during our customer team meeting. With these requirements , we were able to generate the abstract and concrete architecture , helping us bridge the communication between stakeholder and developers . With these architectures, we were able to begin the production of our project.

Belows shows all the new/changed features that we have added to the game in a tabular format . We have chosen this format to make it easier to read for the Customer and Stakeholders. The requirement id of the feature is also displayed as well as how that feature is linked to the concrete and abstract architecture that we generated. In addition to this, Justification as to why we added this feature is also displayed and the relevant class names that have a role in the feature being implemented:

Feature	Requirement _id	Architecture and how it is linked	Change/New feature & Justification	Relevant class names in code
Difficulty Level	UR_Difficulty_ Level	This feature links to both the abstract (specifically the behavioural diagram) and concrete architecture. The feature is a multiplier which works by increasing the number of obstacles in all lanes of the game. As each round progresses in the game ,the number of different obstacles also increases as well. This feature relates specifically to the class ChooseDifficultyLevel in the behavioural diagram .	This a new feature in the game as it is an implementation of a new requirement	Lane - Number of obstacles present in each lane is dependent on the difficulty level chosen at the start of the game . A hard difficulty will have a higher number of obstacles as opposed to the easy difficult level having fewer obstacles. This is done in the updateRound() method.
Power-Up	UR_Power_Ups	This feature links to both the abstract and concrete architecture. In the state machine architecture , it is presented as a power-up detected class which shows that when the player boat collides with an object specifically a power-up object , it is detected and the specific abilities that the power-ups have are applied to the Player boats attribute. In the behavioural class diagram , the PowerUp class relates back to the GameScreen class . For the concrete architecture,the power-ups relate back to the PowerUp class . When the player boat collies with the power-ups , the player will gain new abilities such as a speed boost, agility boost, stamina pack, health pack, 5 seconds of invincibility and a thing that takes 5	This a new feature in the game as it is an implementation of a new requirement	Power-Ups are implemented in the PowerUp class, in which they act as a collidable entity, similar to Obstacles. Their type and stats are denoted by the PowerUpType Enum, and a player gains the Power-Up by colliding their boat with the PowerUp entity.

		seconds off of your time		
Save	UR_Save_Game	This feature links to both the abstract and concrete architecture. In the state machine architecture , it is presented as a Save game class and Load game. In the behavioural class diagram , the SaveGame class relates back to the MainMenu class . For the concrete architecture,the save game feature is implemented as a type in the Boat, PowerUp, Obstacle and Lane class. The save game feature is implemented as follows: There are 3 save slots where the player can save the game by which can be done via the F1, F2 and F3 button whilst the game is in progress . This will save all the appropriate features such as the position of the player boat , the current leg etc. The game is then loaded up via the boat selection screen through the same buttons depending on the specific slot. So F1 will load the first game save slot.	This a new feature in the game as it is an implementation of a new requirement	The classes that implement the save feature are: Boat, PowerUp, Obstacle and Lane. Each of these classes has a save() method which is called in Lane. All of their necessary attribute values are written to a JSON file; with each save slot having its own file. A player can save the game by pressing F1, F2 or F3 for each save slot. These attributes are then loaded into the game during the BoatChoice screen from which a constructor for each relevant class will read the JSON string.
Implementation of Lane Class	UR_Race_Track	N/A	New feature, more possibilities for testing and more control over lanes. Relates back to an old requirement.	By having a class to manage each individual Lane, it makes scalability easier in terms of adding or decreasing lanes. Rather than rendering each Boat, Obstacle, and Power-Up in one central render function, the presence of the Lane class allows for better modularity as each Lane has its own Boat, Obstacles and Power-Ups.
Implementation to hitbox system	UR_Damage FR_Damage	N/A	New feature, replacing previous collision detection mechanism	Moved collision detection to its own EntityHitbox class and built new hitboxes.Previous system was primitive , as it relied purely on the image sizes of the boat. This led many to problems where the images of the boats had whitespaces . This meant that the boat would have a larger hitbox than intended so when the boat passed

				an obstacle but the white space was within the vicinity of the obstacle , they still took damage. This feature was fixed.
Refactoring of code to allow unit testing	N/A	N/A	Change, allows for easier implementation of Unit Testing.	Entities have their textures loaded by a separate method rather than in the constructor. Each Boat, PowerUp and Obstacle object has their own texture rather than using the texture from their Enum (BoatType, ObstacleType and PowerUpType, respectively). This allows the objects to be tested without graphical rendering.
Change how the background is stored and rendered.	N/A	N/A	Change, more efficient loading of images	The names of the background files were previously being searched for and loaded, which led to crashes. Changing this to loading each file in the background directory makes this process more efficient and rigid.
Boat AI	N/A	N/A	Change to CPUBoat	Added hitbox in front of CPU Boats in which if an obstacle collides with that hitbox, it decides if the obstacle is to the left or right of the boat, and the boat will move to avoid it. This change was made to give the sense of smarter CPUs.