The main association is between the stage and “entities” which are created by the stage at appropriate times and in appropriate quantities, hence why the stage will have to keep track of how many obstacles are currently present so it is able to respond to a deficit.

The finished program will consist of roughly two types of entities, hence why they have been split into two on the UML diagram, The first type is “Boat” and the second “Obstacle”, they both have in common the attributes position and size which describe where they are shown on the screen as well as being used to determine when a collision happens between two entities.

Every boat on the stage will have a value for current speed and health as well as statistics specific to each type of boat which may have an effect on the two former attributes. The other attribute, exhaustion, is increased sequentially after each leg of the race and will determine a decrease in certain stats. The boat needs to be able to process a collision with an obstacle (damage to health as well as slowing the boat’s movement) hence the function takeDamage(), the other function move() will be used to increase the speed of the boat.

Boats can be divided again by an important distinction; whether they are controlled by the player or by the program’s AI. There are not necessarily separate attributes for these two different types of boats but they are each involved in contradictory systems. The player controlled boat operates using an input based system whereas the enemy uses AI controls which chooses the best next action by looking for impending obstacles to avoid them. Both are affected by the collider system which is always checking for a collision between boat and obstacle so that it can cause the affected boat to decelerate.

The obstacle entity doesn’t contain anywhere near as much information, instead it simply contains the description of how much damage it will do should one of the boats collide with it. There is also the possibility of the obstacle being able to move itself (for example, a duck) in which case it will simply move between points while avoiding other obstacles.

The sequence diagram shows how the program starts by creating a certain amount of obstacles that is appropriate for the current stage (i.e more obstacles on later stages), then immediately begins checking for any collisions that may happen between boats and obstacles. When this happens, the appropriate damage/penalty is sent to the boat while a new obstacle is requested to be created to replace the obstacle that was previously hit.

When the boat reaches the end of the finish line and assuming it is on >0 health, the program will tell it how many have passed the finish before them and thus how many points they should get for their ending position in the race.

The concrete representation develops the initial diagram to include an implementation based view on the program classes. This representation will make it easier to implement when it comes to the actual programming as the syntax is in pseudocode and attributes are more detailed.

One example of the development of attributes to make implementation easier is the position and size in Entity, which are now stored as two float values each, this is because describing the position / size in 2D requires two values and trying to store two values in one attribute

needlessly overcomplicates the programming by requiring an abstract data type. Stored as two values each, the overall position/size can easily be deciphered by taking both the values. Another new element that is specific to the implementation is the inclusion of the sprite attribute which contains the image that will represent the entity on the stage.

For similar reasons as above, other attributes like: speed, stats and exhaustion have been split into multiple attributes to simplify the implementation. As well as this, the Stage object now stores more information on what specific types of obstacles are present so that it is able to create new ones of the correct type.

It's worth noting here the new obstacle type: DynamicObstacle. This is a unique feature of our program whereby certain obstacles will be created, land in a certain spot and slowly sink into the water (An example could be a falling tree branch). This type of obstacle doesn't move like the moving obstacle but it does need to choose a random spot to land at, know how long it will be in the stage for and how often the obstacle will occur.

There is also a specific archetype for a stationary obstacle because although it doesn't move around on purpose, we will include a passive rate at which the obstacle will flow down the river with the current, hence the attribute "passiveWaterCurrent".

There are also methods in the Stage object which set a precedent for what happens when each race is over, whether because the player has crashed (when the Stage would use the resetToMenu() function) or because they made it to the finish line (nextStage()). The method move() from the original representation is also split into acceleration and rotation which will be two distinct functions in the implementation; we don't want the boat to be able to move in unnatural ways, such as going backwards or perpendicular to the direction the boat is facing.

The AI-controlled EnemyBoat allows us to meet the requirement U2: User races against other boats. The nextStage() method as well as currentStage allow the game to transition through the four races as required by U3

Points are given based on the position of each boat at the end of the race, as shown in the sequence diagram, and stored as boatScore. This can then be checked before the third iteration of nextStage() to check whether the player should be in the final race. This allows us to meet the requirements of U4.

The Stage will create obstacles appropriate for the currentStage value. The collider will then be ready to detect if the obstacle hits a boat as well as if the boat leaves it's lane or hits a river bank as was laid on in requirements U6 and U7. As the boat is required to decrease in health from such a collision (U13), the collider can trigger the decreaseHealth() function. Along with the collider, the PlayerBoat is also controlled by keyboard inputs in the Input system and four associated key presses in order to meet U8.

Each stage will have a value for the max obstacles (stationaryObstacles, movingObstacles and dynamicObstacles) with the values increasing for each subsequent stage. This is because the game is required to become harder with each leg (U15). The stages will also increase in difficulty due to the increase in exhaustion after each leg.