Implementation

When we adopted Undecided's project for Assessment 3, we first looked at the overall project brief for Cohort B to see what we had to implement to provide an acceptable submission for the assessment. Following this, we looked at Undecided's updated list of requirements, and made a detailed comparison of these to the code we had adopted. After a team discussion, we decided that we did not need to amend this list further, as the requirements were sensible and not over-ambitious, however as we progressed, we realised that we had to make some minor changes to the requirements that weren't likely to affect the player experience [1].

Based upon these considerations, we identified two main areas to work on - namely, the minigame, and adding the full number of departments and colleges. We then split our team into two groups of three, each half working on one of these two areas. Once we had made significant progress on these areas (i.e. a functional implementation), individual members were assigned and worked on the smaller aspects of fulfilling the requirements.

We decided to continue Undecided's project due to their well laid out assessment deliverables and code, as well as a list of requirements that wouldn't be difficult to implement given the short amount of time available for this assessment.

To provide a representation of the concrete architecture as updated, we have used IntelliJ to generate a UML diagram [2].

New and updated features

Feature: Departments and colleges

In order to implement requirement F5, we have updated the College class to include Goodricke and Langwith, and we have updated the PirateGame class to include ComputerScience as an instance of Department. This provides a "Campus East" portion for the game, fulfilling requirement F1.1. The ShipType class has been updated to allow ships to be affiliated to one of the five colleges. SailingScreen has been updated in order to return the affiliation of a particular ship, as well as to provide for the physics collisions of the new departments and colleges.

Colleges, once captured, now allow the player to purchase temporary buffs, fulfilling requirement F8.5. These are defined in the new BuffConstant class.

Likewise, departments now allow a permanent stat boost to be purchased. DepartmentScreen has been updated accordingly. This fulfills requirement F9.1.

Justification

We included more colleges and departments in order to meet the requirements set out by the customer in the beginning of Assessment 1 as well as allowing us to separate the map in two. These changes had no impact on the rest of the game apart from requiring us to figure out a way to ensure players know that the new area is significantly more difficult than the first.

Sea of Geese

Sea Monsters

Sea Monsters have been added as a random spawn chance in neutral areas, finishing the implementation of requirement F3.2. Sea monsters are stronger than, but half as common as, randomly encountered enemy ships. This allows for some variety in the enemies the player encounters.

Justification

We felt that just having enemy ships would make for a fairly repetitive game way back in Assessment 1 and Undecided had the same idea in their requirements. We decided to make the sea monsters rarer but also much stronger than regular ships which helps to present them as a much more challenging and fearsome enemy to encounter. This helps to make the progression for players more tangible as eventually they will be able to easily defeat these enemies towards the end of the game, making them feel more powerful.

Feature: Game progression

The SailingScreen class has been updated to add an “air wall” preventing the player from accessing the Campus East potion of the game before they have defeated James and Vanbrugh colleges. This allows for a sense of progression in the game, as well as providing two distinct areas of the map, contributing to the implementation of requirement F1.1.

Justification

We decided that this was the best method to limit the player’s progression and make it obvious that the player shouldn’t go into this area yet. We discussed making it so that the enemies in the second area are much more difficult than the first area, but we found that this would make for a frustrating experience for the player, as they wouldn’t have any idea that this area was supposed to be more difficult, die and then lose all their progression so far. As a result of this, we realised that a more tangible blocker such as an invisible wall would make it much more obvious to the user that they had to do something before they could access this part of the game. Doubling down on this transparency, we included a HUD notification so that when the user touches the wall, it says that they must defeat James and Vanbrugh to progress.

Win condition

The game is now won if all colleges are captured, fulfilling requirement F2. This is implemented via the new WinScreen class that displays the winning message. Requirement F10 specifies that there will be a final boss that will be strong enough that the player will need to have a certain level of upgrades in order to defeat them. While we do not have a specific final boss, Langwith college is strong enough to necessitate player upgrades, and therefore we feel that we have implemented this requirement.

Justification

We wanted it to be rewarding for the player to defeat the final boss and so we decided that a winning screen would have that effect. Once the player has defeated all of the departments, the winning screen congratulates you on winning. We thought that it would be easier to give players the option of being able to go for the harder bosses of each zone first, and so we decided to make it so that you win after defeating everything instead of simply defeating a final boss. This also prevents the user from getting very lucky with the enemy accuracy and being able to win straight away through sheer luck.

Sea of Geese

Weather

A weather system has been implemented using the Weather class. This implements requirement F3.3. Weather is either normal or stormy. Sailing through stormy weather increases the point yield to 2 per second, but also gives 1 damage per second. This implements requirement F6.3. The ifUpdate() method selects a region of the map at random every 30 seconds. This region then has its weather updated at random every 10 seconds. This allows for relatively unpredictable weather. SailingScreen has been updated in order to pass information about the weather to the screen.

Justification

We decided to only stick with two weather types (normal or stormy) as we felt that too many would end up being too confusing for the player, as the only way they would be able to understand what each one did would be to play the game for a long time and observe their health and gold etc for each different weather condition. Having the stormy simply hurt the player also makes it much easier for the player to understand that they should avoid the bad weather, whilst the increase in points being given gives the player a unique risk/reward choice.

Minigame

This fulfills requirement F12 by adding a minigame that is separate from the main gameplay. The minigame allows the player to bet 100 gold on the outcome of a race between four geese. Geese are represented by the MinigameGoose class. The MiniGame class produces a list of the geese and randomly assigns each a speed between 1 and 10, thus ensuring the random nature of the game. The MinigameScreen class then handles the interaction between the player and the minigame. If the player wins the bet, they get back double the gold that was bet; otherwise, they lose that gold.

Justification

We wanted the minigame to feel thematically coherent within the game that we were making, and so we decided to incorporate the theme of water into it. If the game was too boring or didn't reward the player with enough gold then it wouldn't be chosen by the player, but again, if the reward was too easy to get then it would make it very easily repeatable by players to no cost. We believed that racing geese across the water made for an easy solution to both of these problems, as we could simply add or remove geese to change the chance of the player winning whilst also allowing them to see the race actually going on in front of them, giving great feedback. We decided to only do speeds between 1 and 10 because we didn't want the player feeling like it was too difficult to win - if the speed of the winner was close to that of the player's it might feel like the player almost won, making them more likely to play again.

Minor changes

The SailingScreen and CombatScreen classes have received minor changes to make them more modular, and thus, easier to test.

Justification

These changes are very small and don't make any difference to the execution of the code, simply to make it easier to both understand for anyone that looks at our code in the future and also to test what we have implemented since taking over from Undecided.

Unimplemented features

Points on the Win Screen

In their requirement specification [1], Undecided said that they wanted the points earned to be shown on the Win Screen. This is because they wanted this to show to the player how well they had done in the game, potentially compared to other people. However, we didn't implement this as we were unsure whether we should make winning the game the success itself or whether to do make it more of a competition due to the points being shown. In the end, due to the limited time schedule and the difficulty of making the points work and showable to the players after winning, we decided to not implement this and instead focus less on the point system as a whole. As a result, we also decided to remove the 'snowballing' effect from staying alive for a long period of time, as this would promote a playstyle that we didn't want to encourage where the user would just stay still to avoid conflict and get a high score.

Bibliography

[1] Updated requirements. https://davidjnorman.github.io/SEPR/assessment_3/Req3.pdf
[2] Concrete UML diagram. https://davidjnorman.github.io/SEPR/assessment_3/uml3.png