Group 12 Report

Members: Sahil Chand, Jonathan Peters, Max Bodifee, Dylan Cantafio

The name of our game is Racoon Rush, a 2D-style arcade adventure game. The protagonist is a Racoon named Rusty and the objective of the game is to go from one side of the SFU's Burnaby campus to the other while fending off a variety of enemies.

Plan:

Milestone 1: For the halfway deadline, we want to have the player be able to move around and eat a donut. To get this done, we will need to get the player sprite done as well as the animations. Secondly, we will need to have collision figured out as we want the player to be able to eat the donut, increase the score, and then remove the donut from the tile. This a simple yet realistic halfway deadline goal and it sets the tone for the rest of the project after.

Milestone 2: We want the game to be implemented by this time. We have a planning board on Trello and it has a list of features that need to be finished. We have been talking on Discord almost every day and this is to communicate our progress and see if anyone needs help. The main things that need to be done are implementing the full map design, adding sound effects, adding all the other entities and having a working scoreboard.

Approach:

To win the game, Rusty will need to collect all of the donuts in the level. Eating these donuts will also increase Rusty's score, which is pivotal to completing each level. Punishments can be found on each level and they will lower Rusty's overall score, and the moving enemies will end Rusty's reign if they catch him. The punishments are leftover food, and the enemies are a rival raccoon gang.

For this project, we decided to use the swing library. We decided to use swing as we need the library to use Maven and we thought it would be a good fit. We also thought that swing would be great for this project and it helped us build a quality GUI.

In our first group meeting for phase 2, everyone got the project set up on their respective systems and we began brainstorming on ways to tackle this project. Max in particular helped get everyone set up and we did this by sharing our screens and Max would tell us the steps to get the system on our systems. We decided on a system where before implementing a feature, we would need to create a pull request and then someone else in the group can view the code before the change is committed.

As for level design, as in our initial plans, we were planning on dividing SFU's Burnaby campus into different places and this is where the levels will take place. We are planning on having 3 main levels and the locations are the AQ, the dining hall and the library. We chose these areas because we all thought that the AQ and the Library are reminiscent of mazes and we thought it would greatly fit the theme of this project. Rusty will start in the AQ for the first level and then the next level would be the library. The hardest level will be the Dining Hall and it will also be the final level, as there will be a final boss that will need to be dealt with to finish the game. However, due to a time constraint, we decided to have one quality level that has all of the functionalities that we wanted, as opposed to having three mediocre levels.

The sound effects will be found online and then edited using Audacity. An example of this was taking a chewing sound from online and then amplifying the sound. Then take a burping sound amplifying it and stretching it out to sound longer. These two sounds are then combined and then we have the sound for eating leftovers.

Implementation:

It'll be hard to summarise our entire code in limited space, so we'll just touch on the really important areas. The resource folder has all of our art and sounds, it even has the menu and map. The entity package has the code for collectibles like the rewards and punishments. This is where we set the score for each of the entities as well as the sound effects that come from collecting a collectible.

In our game package, this is where the majority of the code is. We have a sound class that holds each sound in an array and we can access that sound using one of our implemented methods. We thought this was an easy way to try out different sounds as all we have to do to add a new sound is add it into the array and then change the index. We have a collision detector class to help deal with hitboxes, and this helped us work with picking things up and colliding with walls with ease.

Our game panel class is the main panel for our game. This class has a lot of functionalities, such as loading our map, starting the game, playing the sound effects and much more.

Adjustments:

At the time of making the use cases and class diagrams, we wanted the reward to specifically be Timbits. Since our game takes place at SFU, we thought it would be cool to shout out the Tim Hortons on campus. However, we changed the reward from specifically Timbits to just donuts. We did this because we realized that timbits looked just like marbles in our game. We wanted the player to be able to tell that the rewards were food, so we changed the design to a pink donut. Now it is very easy to see what the reward is.

We also changed the sound that is played if the player eats leftovers(punishment). Initially, we had it sounding similar to the donut-eating sound. However, we thought it would be a good idea to differentiate these sounds as they serve completely different purposes. Now when the player eats a donut, a simple biting sound is played. When the player eats leftovers, there is an eating sound followed by a disgusting burp. This is a simple yet effective way to differentiate these sounds and we are very proud of our decision.

Due to time constraints, instead of having 3 levels, we decided to have just one large level as we didn't want to have three mediocre levels. Instead, we focused on designing one big level that showcases all of our desired functionalities. We did this to preserve the overall quality of the game as we didn't want the game to drag on just for the sake of length. Having multiple levels requires a lot more resources and careful planning, as well as finding new ways to keep the gameplay fresh and not repetitive.

Challenges:

A challenge we faced with level design was trying to find the best place to end each level. For the AQ level, we initially had the donuts surrounding the maze but then we thought that it might be better to have a donut in the centre. We also found it quite tough to find good places for both the punishments and donuts. We want them to be mixed up pretty well so that the player has to think twice before picking them up. To tackle this issue, Sahil designed level 1 to have enemies guard both punishments and donuts. This way, it is more unpredictable and it will allow the player to fully explore the map instead of only searching areas guarded by enemies.

Another issue we faced was with using branches, a lot of us were unfamiliar with the whole process and this led to some minor merge conflicts. We combated this issue by distributing tasks on Trello, which is a software where you can write up a task and then a member of the team can write their name next to it, which indicates that they are working on it. There was a minor issue where someone's branch accidentally got deleted, but we were able to recover and now we are extra careful to make sure something like that doesn’t happen again.

When we were implementing the scoreboard, we initially had a main handle displaying the scoreboard and new windows. To combat this we tried to have the GamePanel class handle the scoreboard but we found this to be a way harder task. We discovered that we would need a sequence of parent panels for both the Scoreboard and GamePanel classes for our plan to work, and we treated this as a last resort. A small fix we made was to disable the scoreboard when we open the menu, this allowed us to avoid the whole parent panel scenario while not affecting the functionality of the game.

When we were adding in the sound effects, we initially wanted there to be footsteps. However, we quickly noticed that the footsteps would sound very weird as the player moved around and it was very hard on the ears. Sahil was working on this and he decided it would be best to take out footsteps or at least worry about them until the very end. Instead, he focused on adding sound effects for consuming the donuts and eating the leftovers.

Jonathan decided to custom-make almost all the artwork for this game because we wanted the game to have a consistent theme and our game required specific elements, like our raccoon hero. Creating artwork for software is tricky because you have to constantly consider the size and format of each image. You also have to determine a way to handle all your custom sprites and assets in such a way that your graphics library can load and display your assets seamlessly. Like many 2D games, Raccoon Rush uses a tile-based system to display the game graphics. We settled on an original tile size of 16x16 pixels that we scaled up 3 times to display each tile as 48x48 pixels on our displays. This meant that we were limited to 16x16 pixel .png image files to display walls, obstacles, collectables, and characters. Before this project, Jonathan had limited experience creating pixel art but taught himself how to draw and animate in a pixel art style. The raccoon character creation was difficult, as the main character needed multiple poses when travelling up, down, left and right. This required the accurate reproduction of the correct character proportions and animations for all angles. To create the spinning donut animation, Jonathan used the 3D program Blender to model, animate, and render a spinning donut image that he rendered at the 16x16 tile size.

However, using tiles to display all aspects of the game was quickly identified as a problem! Using tiles to cover the entire map meant that it was difficult to decorate the map efficiently. We wanted there to be some variation in the textures of the map, and achieving this look with tiles would have required many different tiles, each with its specific placement on the 32x32 tile world map. To create the background imagery, Jonathan calculated that ¼ of the map is a size of 768x768 pixels. He used a consistent tiled pattern across the entire map to aid with player movement and visualization. Creating 4 different background maps allowed the painting of grass textures, and water textures where appropriate, and painting shadows for certain elements, like tree obstacles.

Designing the world map was a challenge that required the development of custom tools. The world map was loaded from a text file containing various numbers that corresponded to the tile type, like a wall, tree, collectable, or otherwise. Jonathan quickly realized that designing a maze by editing a text file was a time-consuming and arduous task. He created a simple Python script that loaded a 32x32 RGB image and converted it to the text file format that our MapLoader class could parse. This enabled faster editing and visualization of our world map in a human-understandable way.

Management:

All group members contributed code to the project. Max created the core functionality for the game with initial inspiration from a YouTube playlist by RyiSnow. A lot of work was done to improve the code from the playlist since it didn't follow many of the design and code quality principles introduced in the course. Jonathan created all of the art, coming up with initial designs and constantly improving assets, elevating the quality of our project. Jon also created the main menu. Sahil designed and implemented sound, made this report, and designed level layouts. Jonathan also helped edit the initial map design that Sahil came up with. Dylan implemented the scoreboard, unifying the completion of the project. We found this to be a good distribution of work where people could play to their strengths, and everyone still participated in the actual coding of the game.

When someone wanted help, they would just type in the discord what the problem was and all the other members were willing to help. We found this to be a great method as we were able to solve issues very fast due to the diligence of everyone.

Enhancing Code:

The way we approached this project was that whenever someone wanted to work on a feature, they would create a new branch and do the coding in that branch. When the feature was implemented, the person would then create a pull request and at least two other members of our team would review the code. We'll leave comments and then if everything is good, we will merge the branch. This way we can get multiple opinions on whether the code is great quality and the comments left by fellow team members help us solidify a solid foundation of the expected code quality.

We also tried our best to have good-quality code, and we did this by following some object-oriented design principles. For example, we redesigned tiles using polymorphism. We also had a variety of methods that usually solved one problem each, this led to less bloated code and it is much easier to understand.

Sources:

https://www.youtube.com/playlist?list=PL_QPQmz5C6WUF-pOQDsbsKbaBZqXj4qSq
https://freesound.org/people/TRP/sounds/616622/
https://freesound.org/
https://pixabay.com/music/search/16bit/