Peter Baird, Maggie Drew, and Bobby Innes-Gold
COSC-112
May 5, 2020

Grade

Functionality:

Our game only contains four possible keyboard inputs: “a” and “d” to move left and right, “g” to start the game, and “m” to return to the menu. Our program is very robust to these inputs. Pressing the movement keys results in smooth movement of the mammoth, and the game can only be called up by pressing “g” when the main menu is open. We do have a graphical glitch that occurs when generating platforms. An error appears in the command window and the game lags a bit when this happens. For this reason, new platforms are generated in chunks of 100 rather than continuously to reduce the frequency of this error. We have also added in error messages when uploading all .png image files into the game, which tells the user if the file has not been found or read.

Design:

Our program is well designed and broken down into different subclasses for specificity. Our game was complicated to implement in that both the character and the background have to move. We could not make a static environment in which the only character moves, nor could we implement a system in which the background simply moves around the character like in some classic games. Instead, we had to combine both. We achieved this by including all obstacles, platforms, and the mammoth in a “MovingThing” class. This class was divided into two parts: things that the player controlled and generally moved upward (the mammoth) and things the player did not control that generally moved downward (the platforms). Instead of creating a seperate moving thing for the monster, we included it as a platform which generally moved downward. We implemented a height limit for the mammoth so that it could not move off screen. The only time in which the mammoth might reach this limit is when the platforms are already moving downward, so the illusion of upward movement is maintained. The greatest design challenge in this system was the interaction between these two main types of objects. We designed this interaction by separating the instance of interaction from what happens when interaction occurs. The overarching “World” class determines whether an interaction has occurred, and then the objects themselves indicate what should happen in this interaction. (This process is more thoroughly detailed in the “Sophistication” section)

Creativity:

While we based our game off of Doodle Jump, we added on several specific features that extend the original gameplay and fit in with our broader theme. We kept the core structure of the game intact where the user controls the L/R movement of a jumping avatar (in this case, we

swapped the typical Doodle Jump figure for a mammoth) that jumps on platforms to get as high up as possible while avoiding broken platforms and monsters. We added on a scoring system, where the mammoth earns points by jumping on GrassPlatforms and collecting plants. We also added on a temperature bar on the right side that gradually increases until the mammoth jumps on a snow platform and collects a snowflake. If the temperature bar maxes out before the mammoth collects a snowflake, then the game is over. The monsters are also designed specifically to look like scary purple cows. We provided ample creative elements beyond the basic structure of the game, and therefore are strong in the originality category.

Sophistication:

Our project is sophisticated in that it contains many different objects which are unique in their behavior, and sometimes change behavior or appearance after coming into contact with the mammoth. To reduce this complexity each object keeps track of its own behavior and contains a conditional behavior if it comes in contact with the mammoth. To avoid repetition of the conditions for contact, the mammoth contains a "contact" which can be called to determine if the mammoth is in contact with a platform. The "world" class essentially acts as an overarching supervisor of all these parts, generating them, removing them, and directing their behavior. For example, to allow the mammoth to move higher on the platforms, it first checks the mammoth's contact method to see if contact with a platform has been made. If so, the world calls upon the platform's contact function to see what should happen when the mammoth bounces on the platform. If the platform allows the mammoth to bounce higher, (i.e it is not a broken platform) then the world calls upon all the platforms so move downward a certain distance.

Broadness:

We covered boxes 1, 2, 3, 5, 6 and 7. This is how we incorporated each one:

1. Java libraries we haven't seen in class - To upload the .png and .ttf files we used to create custom images and a special font, we made use of java imports like java.awt.IOImage and java.awt.IOException to upload, store, and draw each image in our program.
2. Subclassing - We created a file called MovingThing.java based off of our work in previous labs working with spheres to set acceleration, position, and velocity for all moving components in our game, and to reset/flip as needed. All items in our game that move down the screen or jump (all platform types, mammoth, monster) are subclasses of MovingThing. In addition, we created several child classes (GrassPlatform, SnowPlatform, BrokenPlatform, and MovingPlatform) that all extend off of the parent Platform class - each has its own special contact and movement function, and may also implement an interface.
3. Interfaces - We created an interface called Collectable.java that is implemented in GrassPlatform.java and SnowPlatform.java to control how the mammoth collects

either plants or snowflakes by landing on special platforms. Collectable has a collect method which sets what happens when the mammoth lands on a snow platform (resets the temperature) or grass platform (adds to total score). Also, Collectable includes a drawItem method that calls the imported image file from Main and a removeItem file that gets rid of the item if it is collected.

4. User-defined data structures - did not incorporate
5. Built in data structures - For storing the monsters and platforms of all types that we created, we first used an arraylist of type Platform which was later changed to a vector of type Platform instead. Using an arraylist/vector made it easier for us to hold multiple types of objects, add and remove elements at any position, and not have to worry about having a sized array.
6. File input/output - To make the graphics, we created several .png image files using a pixel art tool and uploaded/stored/drew the files in our game. The user does not import or export files, but our game does use files that we the creators have already included in the project folder. We also downloaded a version of the DoodleJump font from LimaSky (the makers of the original Doodle Jump game) that can be accessed at http://2ttf.com/9L8ZEUpWnuu
7. Randomization - We primarily incorporated randomization in generating platforms, by randomly assigning positions on the screen to ensure that the user is not playing the same configuration over and over. We also used randomization in platform generation by randomly (with some parameters) selecting how many platforms of each type are generated, and how many monsters are generated.
8. Generics - did not incorporate

Code Quality:

We dedicated a lot of time to making sure that our code looked clean at the end of the process, but we could've paid more attention to keeping our various classes and files organized while adding on additional elements to our game. We extensively documented the function of each class in our final prototype with comments on every method. In large classes such as Main, methods are grouped by the overall function they contribute to. The organization of our project files is based around Main as an entry point into the game, with three states - Menu, GameOver, and Game. The Game class instantiates World, which creates and updates the movement of a configuration of platforms and monsters and the mammoth itself. We eliminated all redundant lines from our code to keep it as clean as possible, but we do have some areas that could be expressed more efficiently. Our final product looks and feels as professional as possible given our current working knowledge of java, although there are some small glitches (mainly in the bouncing of the mammoth, which can sometimes act funky at the apex of each jump) that we could've improved.