Project Proposal

15-112 Term Project – Gyanepsaa Singh

# Project Description

**Gravity Wells** is an original game idea that is based on the physics of gravitational interaction. The objective of this game will be to get an object into a gravity well while avoiding collisions with all the other orbiting planets.

The game field will consist of a gravity well with multiple objects orbiting around it. To get the object into the well the player will, by a slingshot-style mechanism, release an object with an angle and velocity such that it orbits into the gravity well without hitting the other objects.

# Similar Projects

As this is an original game, there are not any projects that are closely similar to my planned one. Instead, I looked for somewhat similar projects with some of the features I intend to use, particularly related to components that are new to me.

The first I found is a game called *Orbit* at <http://playwithorbit.com/>. I came across this as ‘Orbit’ is what I originally intended to call my game, so I decided to look it up to ensure that there are no other games by that name, leading me to this. Orbit’s gameplay involves a number of round planets being put into rotation around a series of flat black holes. This game implements a system for player control of the main ‘planet’ in a way that is very close to what I had in mind, i.e., clicking and dragging with a dotted ‘tail’ to represent the trajectory. I would like to implement a similar motion and intend to look closer at this gameplay to see the specifics of the features such as whether the motion slows/speeds and the exact direction of the tail, to try and implement some of these myself. My game will also likely share some features regarding the implementation of gravity with this, as it also involves objects orbiting around others. The rest of my gameplay will be quite different from this game, as mine involves different gravity depths, a single centric gravity well, multiple other non-playable rotating objects with a single playable one, and spiralling into the centre.

The second similar project I found is a gravity simulator (<https://lab.nationalmedals.org/gravity.php>). While this lacks any form of gameplay, which my project will have, it does involve objects different mass interacting gravitationally – albeit likely a more complex one than I will use. I also liked the graphics they have used, particularly the method of shading, and may draw some inspiration from these graphics for my project.

# Structural Plan

This project will be coded primarily using object-oriented programming.

The rotating objects will all be instances of the *Satellite* class. The term Satellite in this case is used in its technical definition, i.e. a celestial object that is in orbit around another one. Basic physical constants will be defined for this superclass, which will then be divided into subclasses of different types of satellites, for example planets or comets, that would have other properties assigned to them like different densities and different ranges of size to be randomly generated from. This allows for easier organisation of the multiple objects and their interactions, and simpler implementation of the physics.

A second class, *SpaceTime*, will contain the code used to implement gravity and the orbits. This will consists of variables that apply to the entirety of the board, i.e. fundamental physical constants, as well as functions for defining gravitational interaction.

Finally, animation and other interaction will follow the MVC paradigm. Typical animation practices that we used like the timer, redrawAll, mousePressed and keyPressed will be used for user viewing and interaction.

# Algorithmic Plan

The most complex part of this project is likely to be the implementation of gravity.

The main controller of this will be, as stated in the previous section, the *SpaceTime* class. This class will contain variables related to the dynamics of gravitational interaction, primary of which is a ‘gravity map’ that is composed of a dictionary of coordinates as tuples which map to the vector representing gravitational acceleration at that point. There are two possible ways to calculate this map – one is a static calculation at the start of each session which allocates a certain mass to the centre of the gameboard, hence warping spacetime to a level that stays constant until that game session ends. The other is dynamically calculating it based on all the different objects on the gameboard at that point in time. While the latter is more complex and realistic, it will be much more complex and slower, so my aim is to complete the first method before, if I have the time and ability, considering the second, as I feel the remaining features already add enough complexity to meet term project expectations.

The *SpaceTime* class will also contain the functions that are used to implement gravity, which will consist largely of the coding implementations of physics formulae – I anticipate to use mainly Newton’s law of gravitation as well as integration, though Kepler’s laws of planetary motion may also be useful. These functions will be called on the objects in gameplay, and their output stored in the objects as variables again, to be called on each ‘step’ of the game.

Instead of typical variables we have seen previously in this course such as dx & dy, variables will created within the *Satellite* superclass and subclasses for properties of different objects, like density and radius, as well as within the *SpaceTime* subclass for fundamental constants like the gravitational constant (G). These will then be used to calculate properties (variables) for each object such as the mass, weight, velocity, and acceleration, using the functions in the *SpaceTime* class, to be translated later into motion. The benefit of this approach is easier organisation as well as more realistic effects – for example, motion would look unnatural if differently sized objects maintained the same velocity. Care will be taken to ensure variables are hard-coded only for physical properties, and anything that can be derived is done so, to avoid scenarios like larger objects of the same ‘type’ accidentally being assigned lower masses than smaller ones. The level of calculation will, however, likely make the running of the program much slower, so a lot of optimisation is likely to be required – and compromises may end up being necessary to ensure smoother animation and play.

# Timeline Plan

*Week of 14 Nov* – Work exclusively on implementing gravity, i.e. *SpaceTime* class.

***Sun 20 Nov (TP1)*** – Gravity implementation completed – test correctness using single ‘planet’ object with hard-coded radius and mass values.

*Week of 21 Nov* – Work on *Satellite* class, finetune gravitational motion, implement player controls.

***Sun 27 Nov*** *–* Different types of planets completed, player can interact with game.

***Wed 30 Nov (TP2)***

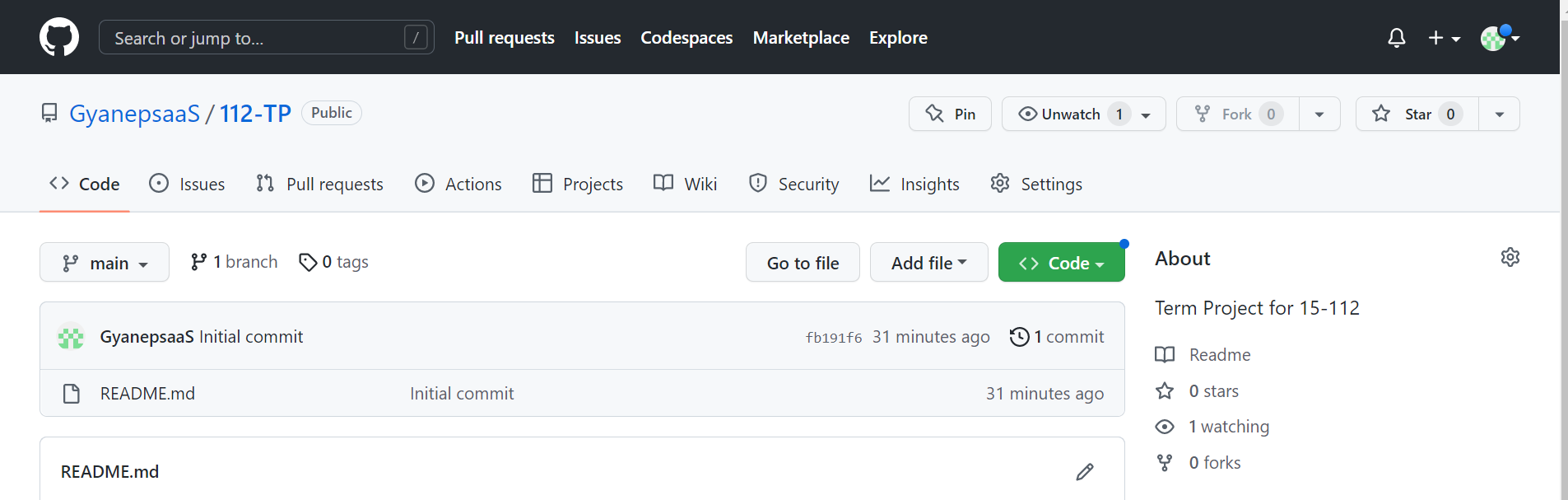
*Week of 28 Nov –* Implement different modes for start screen and gameplay, work on graphics, film video demo.

***Sun 4 Dec* –** Term project completed – buffer time

***Wed 7 Dec (TP3)***

# Version Control Plan

I will use GitHub for version control – I have created a repository for the term project which will contain all relevant files, taking advantage of the inbuilt version control features.



# Module List

No additional modules will be used.

# TP1 Update

I had initially thought that I would implement gravity as a map of forces at different points in space mapped as ‘SpaceTime’. However, I quickly realised that this was a lot of unnecessary calculation for my purposes, and a more effective way would be simply calculating the gravitational force at the points necessary, i.e. the Satellites.

This meant that I also had to consider how I would implement gravity wells. In the process of trying to code gravity, I realised that having gravity wells as a subclass of satellites would be a good way to do this, as they would also have the same properties as satellites in that gravity affects them, but I could initialise parameters like acceleration to zero and add other functions as necessary.

The constants used will also be proportional to real constants such as G, and quantities like mass similarly around the same order of magnitude relative to these constants as real life, but all will be scaled down.

I also initially implemented Euler integration, but upon further research found that instead a second-order integration method may be more appropriate to prevent error.