Project 5
Group 5
12:30 Section

Roshan Chirayil
Jonathon Repta
Michael Ordower
Matthew Searcy

Summary of Project:

Our game will be a multiplayer networked game supporting between 2 and 8 players. The game will take place in 2D, with each player controlling a dinosaur, which they will control using their keyboards. Each dinosaur may walk left, walk right, or jump in order to catch dinosaur treats falling from the sky. Each player will compete against each other to catch the most treats in a finite amount of time. Once the timer runs out, the winner of the round will be decided; the player with the highest number of treats caught wins.

Language and Frameworks:

Depending on the learning curve for Node.js, we anticipate using it in conjunction with JavaFX to design and implement an attractive networked GUI.

We will create our own physics engine. At present, we're expecting to design it in Java, but if the dependence on automated garbage collection slows things down too much, we might switch to C or C++. In this scenario, we might implement some OpenGL as well.

Tentative Classes for Physics Engine:

CollisionBox:

→ contains x, y, width, and height fields, as well as methods to check for collision with other CollisionBoxes

Solid:

→ contains sprite mask and a CollisionBox

Player (child of Solid)**:**

→ extension of Solid containing fields for horizontal and vertical velocity; number of treats collected; a flag denoting whether the player is on a surface on which it can jump; and methods to be called each frame to detect collisions, update position, and accept user input.

DinoTreat:

→ contains sprite mask, CollisionBox, point value, and downward velocity, as well as methods for updating position, detecting whether it has collided with a Solid, and removing the instance from the game.

GameController:

→ object containing lists of all Solids and DinoTreats; a timer; methods for spawning Solids and DinoTreats; methods for updating object parameters each frame; and values for physics settings such as gravity, walking speed, and initial upward velocity after jumping.

Client-Server Interaction:

The server will contain the GameController and will update data at regular intervals. Each client will have a separate GUI application, each of which will contain Keyboard listeners. Each frame, each GUI application will check for Keyboard input and send the corresponding Keyboard info to the server. Once input has been collected from each Player, the server will update the data from its enclosed GameController and then each client GUI instance will receive a copy of the updated list of Solids and DinoTreats, from which it will update the display.

Ideally, we will try to have one machine running as the server and then the rest of the players will be able to remotely connect to that machine. Every player will run the game on their own machine. The challenging part will be sending data from the client to the server but one possible way of doing this is using Serializable and possibly sending game state objects through the socket. We haven't finalized those decisions yet. Depending on how well we pick up Node.js, this might end up being trivial enough.

UI/UX Content Creation:

Sprite design will be handled through the Piskel application, which is available at www.piskelapp.com.

Audio, if implemented, will be created using Ableton Live.