CS-419-SU Group ETA Project Plan

07/01/2016 version 1.0

Table of Contents

Group Name

Eta

Group Members

Megan Fanning fanninme@oregonstate.edu

Brandon Swanson swansonb@oregonstate.edu

Miranda Weldon weldonmi@oregonstate.edu

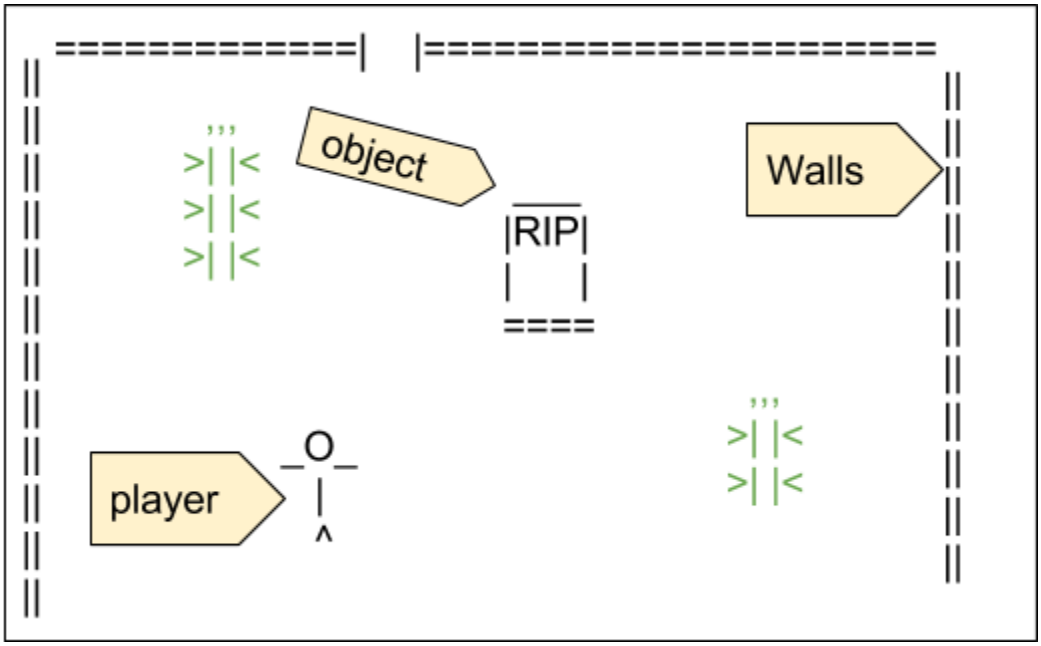
Introduction

Our program is an "endless runner" that procedurally generates obstacles which the users sprite navigates past using their keyboard to move up, down or forward. This program will be networked in such a way that two clients can connect to a server and collaboratively navigate a character through a game world that is hosted on the server and displayed in both client applications.

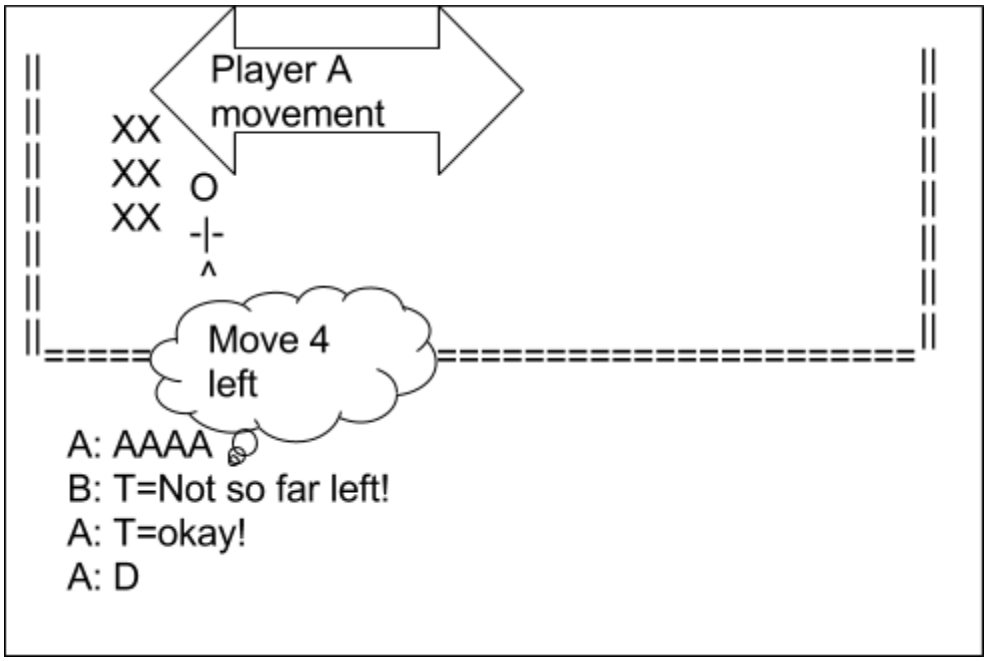
Program Description

Users will run the program in a linux terminal (on Flip server preferably) and move their sprite to navigate past a series of obstacles using their keyboard in cooperation with the other user over network.

The picture below demonstrates what a page might look like.



The view will be top down, players will nativage a dangerous maze (objects they must not hit), this will encourage careful cooperative play where the players should alternate taking turns. Players will be able to enter text to talk to other players and input in discrete units (either up and down or side to side depending on what axis they control)



Structure of the Program

There will be a user sprite object which contains animations of the sprites movement and location on the screen including a hit box to determine if they have successfully avoided the obstacle.

There will be a obstacle class which have the dimensions of the object and location. Additionally there should be a user interface to allow users to start a new game a view high scores.

To facilitate a “graphic” experience for the user and network control it will be necessary to maintain a game state object, maintained by a server program, that receives input from users and correspondingly updates and shares the modified state with clients. An effective development model for this problem would be the Model View Controller strategy. The model contains the “game rules”, the application logic governing how the game state should be generated and affected by the user. The controller is responsible for receiving input from the user and manipulating the model. Finally the view is what the users sees that represents the state of the model.

In our networked endless runner application the model would reside on the server application and the server application would communicate to the clients the necessary information to maintain a view of the game state. The controller classes would reside on both the server and client and would be responsible for communicating user input from client to server and then using that input to manipulate the model. Finally the displaying the state of the game world, the view class, will be handled in the client applications and accomplished using curses library.

Design decisions made thus far:

- Player will move discrete amounts (ie one key stroke equals one unit of movement)
- Each entity on the screen (character and obstacles) will have a x,y position representing its top,left position, an ASCII representation, and a simple hitbox.
- Process loop -> client pressing key, sent to server, server updates game state, check game rules, send updated state of game to client (as JSON array of objects and locations), client renders visuals of game state using curses. Client renders image, server holds gamestate.

Libraries, languages, APIs, development tools

- Program will be written in Python with curses for the graphical and input library.
- The Socket library will be used for networking, During the first weeks of development we will determine if a more heavyweight networking library will be necessary for managing an event queue and object serialization/deserialization such as the Python wrapper for ZeroMQ or using JSON or pickle to to serialize objects.
- Flip server (linux) will be the deployment platform (for both client and server scripts)
- We will use a Github hosted repository for development collaborating and intend to use a master/dev/topic branch strategy similar to this gitflow workflow

Team members tasks and objectives

Megan’s Tasks (Networking)

Hours of work	Task

5	Research networking libraries and methods of serializing object
2	Plan networking design
5	network server & a single client
2	refactoring, debugging, and testing networking of client and server
5	Allow client to send commands
3	Allow client to send a “chat”
2	refactoring, debugging, and testing client commands and chat
15	to set up server to handle multiple clients
10	Handle client control (client A vs client B turn taking etc)
4	refactoring, debugging, and testing handling of two clients and client interactions
5	To properly accept input and properly output user input to functions
6	Implement JSON serialization/formatting of input
3	Collaborate with teammates on input , output and formatting of data
3	Finalization of design including refactoring, debugging, and testing
5	Write up final report
75 total	

Before the mid point goals include:Networking the server and client to send commands. Allow two uses to connect to the server and send commands. After the midpoint of the course objective are to network the server to send JSON formatted files, handle clients, and allow clients to send text to each other and work on the test suite.

Brandon’s Tasks (Curses Input and Output API, client script)

Hours of work	Task

10	Verify functionality of all necessary curses operations of flip server with working prototype of non blocking output and input loops in an ASCII terminal
8	Develop interface for objects to be drawn, defining a pattern for our player character, world objects, and any other other entities to yield a position and collection of ASCII characters and position Possibly use external files for storage of “graphic attributes”, ascii patterns or sequences and color information.
5	Create debug-mode text based human readable view of the game state, for debugging problems and to alleviate the critical-path by ensuring that network and game model development can be tested independent of View functionality
8	Create main process loop for client script that handles non-blocking input from user, output to screen via curses, and sending receiving network information
6	Work with Network Engineer to establish API for sending keyboard input to server and receiving game state rendering information over network, connect those network IO APIs to Curses IO loop
4	Create client initialization routine, determine available canvas (terminal character width and height-1) and report to server, server side will then determine maximum shared window between to client terminals and inform clients of the output dimensions
8	Implement rendering function, receives as a parameter a collection of objects (received over network) that possess sufficient information to render an image of the game world model contained on the server. Handles drawing and coloring of objects as well as resolving conflicts of overlapping objects if overlapping is allowed in game model (ordering strategy to be determined)
2	Define an easily maintainable method for mapping keyboard input to available user actions (it should be trivially easy to replace one control scheme with another and to that end a character value should never be hardcoded in an if statement)
8	Determine viability of “continuous input” vs discrete input over network Strategies discussed are: send keydown and keyup events over network, continually send keypress over network, or send only one event per key. Choose one method through research or prototype and implement input functions accordingly
8	(With Network and Game-Model engineer) determine minimum network message frequency and size needed to render the game world on the client

	size, if possible implement a Delta-Messaging strategy where only the changes in the game state are transmitted. If using deltas then create system for maintaining and updating client side model of the game-state that can be updated via the delta messages and then re-rendered.
3	Prepare midpoint demonstration suite with setup and operation instructions, Prepare a release commit of git repository with stable features
8	Develop graphical embellishments, animated environment and exciting death animation.
8	Create login screen with high-scores (received from server) and ability to choose name and connect to game
5	Work with team on final report
91 Total	

Before midpoint goals: Successfully attach input output loop to network output and input APIs, by the midpoint the client should be able to reliably send keyboard input to the server and receive updated world state from the server and display the updated world state.

After midpoint goals: Implement graphical embellishments and animations, create login/high-score screen, contribute to testing suite

Miranda's Tasks (Game State)

hours	task
5	Open connection to network, start game and quit game (close network connection)
3	Create player sprite (movement, location)
15	Create obstacles (location of obstacles random generation)
20	Create "endless" levels that randomly or procedurally generate obstacles, their placements, and level layouts.
5	Track time and variables for user score
8	Planning and initial coding of client interactions
25	refactoring, debugging, and testing handling of client interactions (movement, hit box, etc)

3	Calculate/Store hi-scores in human readable format
5	Work with team on final report
89 Total	

Before midpoint goals: Gamestate, majority of planning, implementation (sprite creation and animation, movement tracking), objective/high score calculations, initial test rooms and obstacles.

After midpoint goals: Create endless game generation, obstacles (location of obstacles in random generation), sprite object interaction.

Testing Suite (Developed by all team members as time allows)

5	Research methods of Simulating client keyboard input
20	Create pseudo-client that can connect to server as 2 players, controlled by one keyboard or testing input
10	Create test suite (move character up, down, left, right, into objects etc)
5	Use test suite to verify integration and prevent regression after each major feature integration
40 total	

Conclusion

Our Project will allow two player to move a sprite through an endless “maze” of obstacles sharing the movement of the sprite. Users will be able to communicate using a chat command to talk through how to move safely and share the controls to successfully win.
Design features will include

- Networking 2 clients responsible for receiving input from user and rendering the game
- A server responsible for maintaining a game world and updating it according to the input from the client programs
- A game model that implements the rules of the game
- A test suite capable of continuous integration and regression testing

The gameplay will consist of two players attempting to navigate as many screens as possible without colliding with deadly obstacles.