Fructus Victus

Wentworth Institute Of Technology
Department of Computer Science
Software Design and Development COMP 566
Spring 2015
Software Design
Adam Christensen, Zach Thornton, Brian Quinlan, Tony Li, Steven Fortier

1. Introduction

1. Purpose of this Document

The purpose of creating this Software Design Specification (SDS) document is to describe key components and functionality of Fructus Victus.

2. Scope of the Development Project

Fructus Victus is a procedurally generated game which utilizes ray-casting to render a maze that users explore. The three key components of Fructus Victus are the ray-casting game engine, map generator, and input handler. The ray-casting game engine is the component responsible for calculating and rendering the pixels on the screen. The map generator creates randomized maps based on procedural generation algorithms, and the input handler manages keypresses from the user, translating the commands to movements on the screen.

3. Definitions, Acronyms, and Abbreviations

2D - An environment that uses the X and Y axis to display in two dimensions.

3D **-** An environment that uses the X, Y, and Z axis to display in three dimensions.

JRE - Java Runtime Environment

LibGDX - A java game library

Ray-casting - A rendering technique which recomputes the user's visual frame whenever their coordinates or viewing direction changes. For each vertical line displayed on the screen, a ray is casted from the user's position until it collides with a wall. Once the collision has occurred, trigonometry is performed to determine the height of the wall.

SDS - Software Design Specifications

XML - eXtensible Markup Language, a set of rules for formatting text documents that is both human and machine-readable.

4. References

1. Chollak, D. (n.d.). Libgdx/libgdx. Retrieved February 3, 2015, from https://github.com/libgdx/libgdx/wiki/Gradle-and-Eclipse

2. Loftis, H. (n.d.). PlayfulJS. Retrieved February 3, 2015, from http://www.playfuljs.com/a-first-person-engine-in-265-lines/
3. Permadi, F. (n.d.). Ray-Casting Tutorial For Game Development And Other Purposes. Retrieved February 3, 2015, from http://www.permadi.com/tutorial/raycast/index.html
4. Procedural Content Generation Wiki. (n.d.). Retrieved February 3, 2015, from http://pcg.wikidot.com/pcg-algorithm:map-generation
5. Quinlan, B., Fortier, S., Li, T., Thornton, Z., & Christensen, A. (2015). Our SRS. *VICTUS*.

5. Major software requirements

The software requirement for Fructus Victus is an up-to-date version of the Java Runtime Environment.

6. Design constraints, limitations

Map designs must be generated as a two-dimensional array. Users are unable to traverse vertically through the world, nor jump.

6.1 Design Constraints

N/A

6.2 Limitations

Fructus Victus is limited to devices that support the JRE, and is functional under the condition that the user has both a keyboard and mouse.

7. Changes to requirements

7.1 Usability

- The user can launch the game with no prior knowledge.
- The user can change the settings menu by using text descriptions of each button.

7.2 Reliability

- Errors will be handled.

7.3 Robustness

- The game is saved every 5 minutes in case of failure. When game relaunches the user can reload the saved state.

7.4 Availability

- The game runs at 60 frames per second assuming a computer with a minimum of Intel GMA 4500MHD, 4GB RAM, any 2.0GHz dual core processor, and Java.
- The game runs with a minimum video resolution of 1024x768.

- The game is available after installation.
- The game runs on Windows, OSX, and UNIX.

7.5 Maintainability

- The game is open-source and available on GitHub

8. Overview of Document

Part 2: Data Design - This section describes the components required for the game to function.
Part 3: System Architecture Description - This section describes how the game is organized and provides an overview of modules.
Part 4: Detailed Description of Components - This section provides a more in-depth explanation of the modules outlined in the previous section.
Part 5: Interface Design - This section showcases Fructus Victus' user interface and menu system.

2. Data Design

Map (Class)

- The Map class manages Map Chunks. It handles stitching the map together, and issues commands to generate new Map Chunks when the user completes a level.

MapChunk (Class)

- The Map Chunk class contains an internal two dimensional array that is 128x128 in size. An instance of the Map Chunk class has the ability to return it's internal array. When a new instance of Map Chunk is created, it then randomly generates contents based on the other Map Chunks that exist in adjacent positions.

Renderer (Class)

- The Renderer class performs ray-casting operations. Contained within the Renderer class is a render method that casts a ray from the user's origin for each vertical line that will be drawn on the screen. Once the ray-casting operation has been performed, the current line's coordinates are stored in a list, which is used in the Game Main class to draw textures at the appropriate locations.

Game Main (Class)

- The Game Main class acts as a manager for all of the individual components of the game and contains the game loop.

Desktop Launcher (Class)

- The Desktop Launcher class contains platform-specific code that needs to be defined, such as alternate control schemes, or rendering methods.

3. System Architecture Description

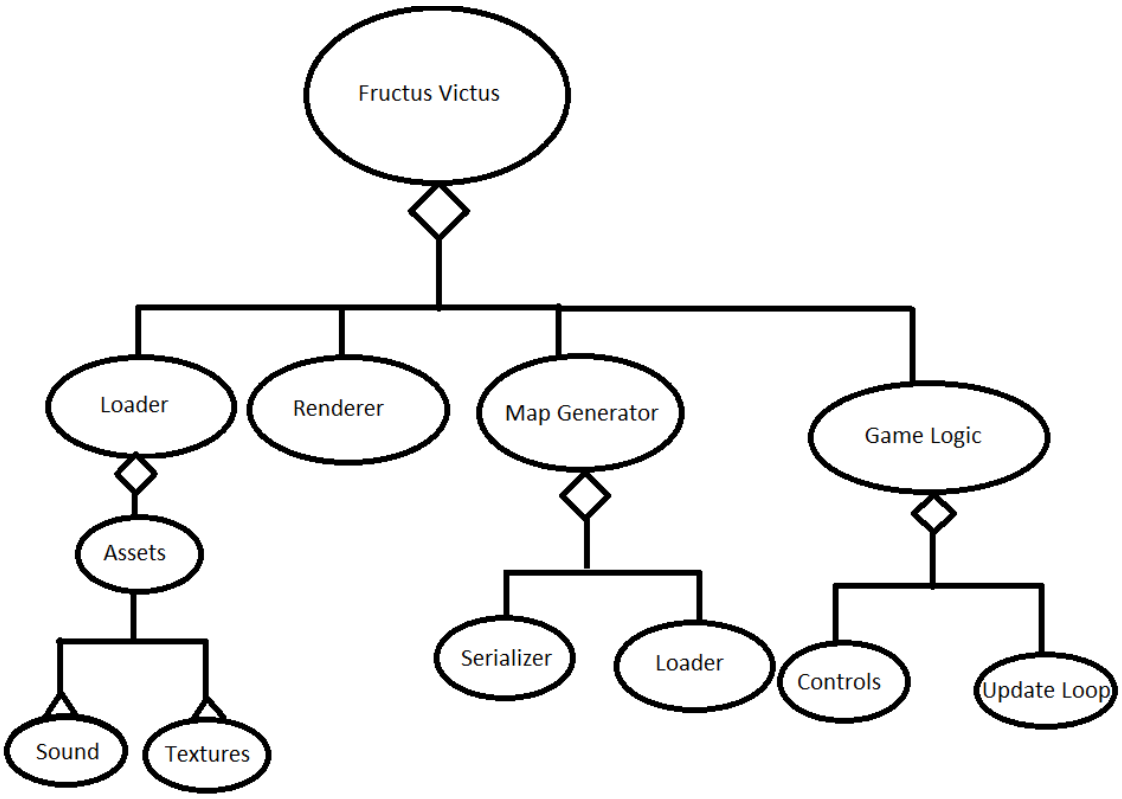
1. Overview of Modules / Components

Raycast Map Renderer: This component contains the method for transforming a simple two dimensional array of numbers into a three dimensional world.

Map Generator: This component contains the method of procedurally generating the two dimensional array that represents the game world.

Input Control Manager: This component manages all incoming user input, and relays information to other components so they can respond accordingly.

2. Structure and relationships



4. Detailed description of components

1. Component template description
    1. Raycast Map Renderer
    2. Map Generator
    3. Input Control Manager
2. Description of Raycast Map Renderer

Identification

Raycast Map Renderer is located in the “Renderer.java” file.

Type

This component is a class.

Purpose

This component renders and updates the images that the user sees.

Function

Renderer:
- Performs ray-cast operation for a vertical line on the screen.
- Determines which specific tile was hit by the casted ray.

Inputs:
- User X Position.
- User Y Position.
- A direction vector, indicating where the user is facing.
- A two dimensional array for the current map chunk that the user is in.

Algorithms Used:
- Ray-casting

Subordinates/ Modules used

Functional Requirements:
- Raycast Map Renderer renders image on screen.
- Raycast Map Renderer handles x and y position changes, x and y direction changes, and x and y plane changes, from input control manager.

Dependencies

This component is dependent upon the map generator supplying the maps for

it to render. It is also dependent on the movement of the user, because ray-casting only re-renders the image when the user's location or direction changes.

Resources

The minimum resources required are an Intel GMA 4500MHD, 4GB of RAM, any 2.0GHz processor, and Java.

Processing

```
Render()
        For(line in screenWidth)
                calculate direction of current ray
                while(ray has not hit a wall)
                        move ray in current direction
                calculate total line height
                calculate line start coordinate
                calculate line end coordinate
                add lineHeight to batch
                add current x coordinate to batch
                add starting y position to batch
                add ending y position to batch
```

Data

Data is stored for save files in XML format. These files indicate the user's progress and automatically update every five minutes.

posX - The users X coordinate, initialized to 22
posY - The users Y coordinate, initialized to 20
dirX - The direction the user is facing vertically, initialized to -1
dirY - The direction the user is facing horizontally, initialized to 0

xBatch - A batch for x coordinates
y1Batch - The start y coordinates
y2Batch - The end y coordinates
lineHeightBatch - The height of the render stripe

3. Description of Map Generator

Identification

Map Generator is located in the "MapChunk.java" file.

Type

The Map Generator is a function within the MapChunk class.

Purpose

The purpose of the Map Generator is to create map chunks with paths that connect to adjacent map chunks.

Function

Generator
- Generates map based on a given height and width.
- Makes sure there is a path across the map to given points.

Inputs:
- Map Height.
- Map Width.

Algorithms Used:
- Map generation.

Subordinates/ Modules used

Functional Requirements:
- Map Generator generates map chunk.
- Map Generator returns map height, map width, and the map to renderer.

Dependencies

The Map Generator relies upon the input handler to know when to generate the next map chunk. The Map Generator is also dependent upon the renderer to display the generated map chunks.

Resources

The minimum resources required are an Intel GMA 4500MHD, 4GB of RAM, any 2.0GHz processor, and Java.

Processing

N/A

Data

mapHeight - The height of the map, initialized at 128
mapWidth - The width of the map, initialized at 128

map - A 2D array to hold the map

4. Description of Input Control Manager

Identification

Input Control Manager is located in the file “Controls.java”

Type

The Input Control Manager is a class

Purpose

The purpose of the Input Control Manager is to make sure that the controls that are being input are relayed to the renderer.

Function

Update
- Repeatedly loops waiting for user input
- Based on input, checks validity and if valid updates user position on the map.
- If paused, a second set of checks will occur relevant to the menu controls

Inputs:
- Keys, supplied by user

Algorithms Used:
- None

Subordinates/ Modules used

Functional Requirements:
- Input Control Manager processes keypresses.
- Input Control Manager updates user position.

Dependencies

The Control Manager is dependent on user input and the map to check for wall collisions.

Resources

To play the game the user is required to have a keyboard.

Processing

```
update()
    if keypressed == key.up and willNotCollide()
            movePlayer(up)
    else if keypressed == key.down and willNotCollide()
            movePlayer(down)
    if keypressed == key.left and willNotCollide()
            rotatePlayer(left)
    else if keypressed == key.right and willNotCollide()
            rotatePlayer(right)
    if keypressed == key.strafeRight and willNotCollide()
            movePlayer(right)
    else if keypressed == key.strafeLeft and willNotCollide()
            movePlayer(left)
```

Data

moveSpeed - The in-game character's movement speed.
rotSpeed - The in-game character's rotational speed.
forwardKey- The current key to be used for forward movement.
backKey - The key to be used for backwards movement.
rotateLeft - The key to be used to rotate to the left.
rotateRight - The key to be used to rotate to the right.
strafeLeft - The key to be used to strafe to the left.
strafeRight - The key to be used to strafe to the right.

5. Interface Design

N/A