Project 2 Design Document

CS447 - Computer Game Design

Ethan Schluze
Sebastian Sanchez
Sasha Demyanik
Christopher Dang

Game Overview:

The idea for the game that our group plans on working on will be an action platformer with destructible environments that is similar in style to Mega Man but incorporates several aspects from a flash game named “Toxic” (found at http://www.nitrome.com/). The mentioned flash game incorporates destructible environments, so that if the players drops a bomb, the radius of the explosion will remove whichever wall, floor or ceiling has come into contact with the blast radius as long as it’s destructible. The main idea of the game is a character navigating through a map towards the end portal while collecting powerups, using platforms and attacking monsters that are along the route. The player will use various types of explosive weapons that can alter terrain.

The way this game will be played is through the use of keystrokes to control the player and the player’s actions. Arrows keys or WASD keys will be used to control the movement of the player. Space bar will be used to drop bombs wherever the player is and switching between bombs can be done using the S key. There will be various types of bombs with specific functions along with a gun that can be shot forward by pressing the X button. Some bombs that have been placed specifically can then be detonated by pressing Z. Other interactions that the player may need is pausing the game with key P and then muting the ingame sounds by pressing M.

With this key setup we plan to have the player move left and right along the platforms as well as jump onto various platforms. The character will also be able to slide down walls and jump off of walls. The weapons that will be used can affect some of the environment and destroy particular walls and floors depending on the blast radius caused by the weapon. These explosions will alter the terrain in such a way that it will affect interaction with the player. If there is a crater created in the floor, then the player will moved along the slope of the crater and not on top of where the floor used to be. Another type of interaction that the player will experience is between the enemies within the game. Each enemy will do damage to the player if the character collides with the enemy entities. Anything within the explosive radius of the bombs that the player places will be affected. This includes the player, enemies and any portions of the destructible environment.

Every visual entity within the game will interact with the player. Bombs will be represented visually along with each respective explosion animation. The character will be illustrated along with each of the enemy entities within the game. Score bonuses will be represented in the game as some form of object for the player to collect.

The environment will have a form of pseudo-gravity that will affect all entities, such as the

player, the bombs, and enemies. The player will start at the beginning of the level and travel over platforms, tunnel through portions of destructible environment and destroy any enemies on the way to the exit. The player will be able to use a few different types of bombs with the possibility of acquiring additional bombs in each level. One particular bomb that may be thrown by the player will be under the influence of gravity and will explode when it collides with some object.

The driving force behind the game is based off of the main objective - for the player to reach the end portal. This makes the game interesting as the player is faced with the challenge of destroying obstacles and killing enemies before he gets killed. The fact that the player is able to destroy the environment around him makes the game interesting as the player is more in control of platforms. Levels in the game will be designed like a puzzle to force the player to make decisions. Hopefully, the player will gain satisfaction from solving these puzzles.

Development Strategy:

We will reusing ideas from previous assignments like the basic structure of building a window, constructing entities and processing them. The hardest part of the project in terms of technicality is implementing the bitmapping collision detection and making it run reasonably fast.

The aspect of development will first be approached by identifying what we want to implement, and what the unknowns and constraints are. Once we have a good understanding of what we want to develop, we will break them down into manageable pieces and assign them to a team member who will be responsible for the completion of a specific task. If we do not know the answer to a problem, we will do research to solve it.

The earliest milestone would be to get the destructible environment working since many of the other features are dependent on it, so it would be best get a static screen prototype of the destructible environment working.

The features that follow after creating the destructible environment include creating the various game entities such as the character, enemies, and weapons / projectiles. Afterwards comes getting interactions between the environment and entities working. In addition, the latter milestones are to develop two separate collision detections and to test them.

The following features such as creating the maps for levels and collecting the art work and transitions between levels are not dependent on any of the other features and do not have to completed in any ordering.

Each team member will be responsible for the tasks assigned to him, and he could be assisted by other members or switch tasks with them. At this point in the project, we have not assigned particular tasks to anyone yet, but when the skeleton of the game is made, then we can start defining roles for each team member.

By the time the alpha due date comes around, we expect to have our technical showpiece(s), our low-bar goals and at least two levels completed.

Technical Showpieces:

The game will utilize a destructible environment. One of the technical hurdles of a destructible environment is to utilize the Slick2d framework to modify the image on the pixel level

and redraw it to the graphics context. It is yet to be decided what type of backbuffer the bitmap would be using, but eventually it would have to be converted into the Image object that Slick2d uses unless there are better methods to draw to the graphics context.

Since the game does not have a static screen camera (in other words a scrolling camera) locating destructibles must be done with respect to their world coordinates, and modified images need to be recreated in a manner that does not bog down the game. Thus, simply checking every pixel in an image to see if it intersects with another object will not suffice.

With a bitmap there is a minor problem with using a (relatively speaking) large map, as that would consume large amounts of memory. There also can be another minor problem with drawing the correct pixel. This is because entities can be placed in continuous positions whereas bitmaps are placed in discrete positions. So rounding the values must be consistent across all levels of drawing.

The other technical hurdle of a destructible environment involves collision detection. Instead of a simple AABB or circle collision detection, the game must also handle collisions on a lower level of granularity - at the pixel level. This can cause problems for the player such as getting stuck on jagged surfaces, so collision handling should allow the player move smoothly across seemingly jagged surfaces.

The pseudo-second technical showpiece will be an exploration of collision detection algorithms. Depending on the amount of moving entities in the game, broad phase collision detection may not be needed. In that case we will do a comparison upon the narrow phase collision detection algorithms.

If we have a substantial number of moving entities in the game say more than forty then for the broad-phase collision detection we have a choice of implementing the sweep-and-prune, grid, quadtree, and r-tree algorithms. Each of these algorithms have their strengths and weaknesses. Some algorithms perform better when entities are grouped closer together or when most entities are static within the game world.

Sweep-and-prune groups elements together by the bounds of each element. These bounds are made with respect to one or more axis. The main disadvantage with sweep-and-prune is using it with a game that has many moving entities. Although sorting can be done in linear time, speculatively, performing many sorts can slow a game down.

Grid refers to grid spatial hashing. Entities in a game are place into node by hashing the entities position (which is usually done simply by division). Grids are easy to implement, and in previous projects grids have performed reasonably well. The main disadvantage with grids is the memory consumption which can be large from using small grid sizes or using a large map. Most implementations of grid requires a bounded map.

Quadtrees are a 4-ary tree. Each node describes a spatial subsection of the world. They are usually defined as by cartesian quadrants. Elements are sorted into the nodes of the quadtree based on the boundaries of each node. Similar to quadtrees are R-Trees. Quadtrees have set boundary sizes with respect to their depth, but R-Trees boundaries are defined by the elements themselves. R-Trees nodes consists of elements that are relatively close to each other. The main disadvantages with trees are their insert and delete operations. In games that have lots of moving entities, reinserting entities into trees can be very costly, so a method that moves an entity in a tree without reinserting must be implemented. Quadtrees are only

recommended for a game world with no bounds.

For the narrow-phase collision detection we have to perform pixel-level detection. So far researching about this topic, there are two methods to pixel level detection. One is the straightforward brute-force method which does a comparison of each and every pixel location that corresponds to the colliding entity. The other method is more practical. It is the collision detection system used by the “Sonic” series of games. The sonic game only compares certain bits that are defined around the entity, so only a subset of pixels are compared to other colliding entities. A more detailed explanation can be found at: http://info.sonicretro.org/Sonic_Physics_Guide. We can however, expand upon that and add modifications to the algorithm to suit our needs.

Low Bar:

The low bar for this game will comprise all of the necessary framework for the game; this includes the bitmapped levels, the collision detection, the sound, and the entities described below. Initially, we want to place a character that will move along a surface with input from the player and have the player jump onto platforms. The character will not be able to go through platforms from underneath but will instead bounce off. Similarly, walls will have an effect on the user and when the character collides with a wall, the player will begin to fall, or if the arrow key in the direction of the wall is held, slide down the wall at a reduced speed.

Another key aspect that must be included in the low bar is the use of screen and world coordinates in order for the map to scroll. This will consist of both vertical and horizontal scrolling as the player moves towards the goal. Enemies and powerups will be rendered as the screen moves closer to their locations so as to lessen resources. The player will have three types of bombs at their disposal; a standard bomb that can be thrown or dropped and detonated by the player, a grenade that explodes and fires shrapnel after a preset timer, and a shaped charge that sticks to and tunnels through the first surface it collides with and is detonated by the player.

The player will be able to use replenishable bombs to affect the environment so that the blast radius of the bombs will remove the section with which it collided. These craters formed by the blasts will affect the player, so that the player does not simply move over the destroyed section, but walks into the crater. The player will also have a gun that he can shoot at enemies with, but the gun will not be effective against some enemies. The gun will have unlimited ammunition but will not be able to damage any structures or entities besides a few types of enemies. The enemies in turn will move around and be able to inflict damage on the player if the character collides with the enemy. There will be two types of enemies: a zombie, which is slow and can be damaged by the gun, and a robot that can only be damaged by a bomb. It is not necessary to destroy all enemies in order to complete the game, the player simply needs to reach the end portal in order to finish the level. A simple scoring system will be implemented where the player will collect randomly placed score objects. If the player runs out of bombs to destroy the environment and gets stuck in the level, then the player can restart the level.

High Bar:

The player will be allowed to crawl under small gaps and run on platforms. Custom graphics will created by us using a camera. A person in a costume will be placed in a white background, and pictures will be taken, then we will crop out the white background using an image processing tool like Photoshop or Gimp and place all the images into a sprite sheet to be able to load it once in our game to create the animation of a character and improve performance as loading different frames of movement of a character has a performance overhead. The game will have more than two levels, and there are going to be additional types of enemies that shoot at the player as he moves. The player is able to spawn one scout per level, which is a little character that is not affected by enemies and the player is able to control it to discover what is ahead in the map. The scout has a timer that lasts 30 seconds, once the timer is up, the scout disappears from the screen.

Conclusion:

In conclusion, this game will bring excitement and entertainment to users as they are challenged with the goal of finding the end portal with the limited resources they have. Many challenges are going to get in the way of the users to keep them from beating the level such as varying enemies and a path towards the exit which is not readily visible to the player from the start of the level. For example, enemies will crop up and will decrease their health level if they come into contact with the player. If health level goes to zero, then the player dies and has to restart the level. This project is challenging enough for our team members to have a significant learning experience and complex enough to be able to complete the low bar in the time estimated. Most of the focus will be on how we implement the various components and collision methods as these seem to be the most complex aspects of our projects. Each feature that we want to include will be divided up, so that the maximum number of features can be completed in the time allotted for this project. With all of these ideas, we hope to provide a game that displays our technical showpieces and actually works.