Brian Bamsch
Michael Gopez
Patrick Thavornkant

CSE 165/ENGR 140
Final Project Report

Project Description:

We decided to make a video game based on Super Bomberman. Super Bomberman is played on a grid-like map where bombs are used to blast away obstacles and, eventually, your opponents. We gathered textures from online sources and the game itself in order to emulate the game as closely as possible.

Members:

Brian was in charge of coding the game's central features and implementing many of the mechanics which go on behind the scenes. Some of these mechanics include: player movement, entity and player animation, texture loading, and NPC AI. Additionally, Brian designed the map/world code and random terrain generation to generate a new environment with each match. Brian also fixed up and debugged most of the errors that existed within all of our respective duties.

Patrick was in charge of handling explosives and their consequential explosions. The concept of bomb drops, limits, blast radius, and flames were the crux of his input. He also took on the role of implementing destructible titles in order to create obstacles for the players. Additionally, he also assisted in gathering music for the game (he asked an alumni of UC Merced, Andrew Tae, to create a tune to play in the background) as well as discussing the concepts for the game's mechanics (such as how things ought to behave).

Michael was in charge of implementing power-ups. He extending destructible objects so that, when destroyed, would have a chance to leave a powerup which a player could then pick up to give them an advantage. These power-ups include increasing the player's bomb supply, increasing the player's bomb blast radius, and increasing the player's speed. Players were then extended to react to these powerups and edit the player's stats based on the power-up which is picked up.

Implementation:

We based the project on the Freeglut support code and also included some freeware libraries to assist in loading image files as textures into OpenGL. There are many bits and pieces that went into implementing the game. However, we can break this into five generalized categories: the Game Window, the World, the Characters, the AI, and the Entities.

The Game Window is the main class that is instantiated when the game is run. The class handles the creation of the World which will be explained in the next paragraph and creation of the players that will be placed within the world. The Game Window also handles resizing, updating, and drawing the game to the graphical window on the screen.

The World is a container for all of the action that is going on in the game. It contains the players, the entities, the world, and a variety of functions which allow all of these objects to interact to form a functional game. When the world's update function is called, each player and entity is given a chance to update (these updates also control the entity's animation) and interact with their current environment. For an AI, this means trying to move towards its current goal or generate a new goal if it has achieved its previous goal. The AI will be explained in more detail later. Like the update function, the world also has a draw function in which it draws a background and each block in the world before also calling the draw function of each entity and player.

The Characters act as the main characters in the game. They consist of playable, human characters, and non-playable AI characters. Currently, we have implemented a 4 character limit mainly because it reflects what is seen in the original game but this could easily be expanded to more players by increasing a variable. All characters receive events from the keyboard and, depending on which player is set to respond to the keys being pressed, their player will move accordingly. NPCs simply ignore these events and handle their movement through the update function.

Characters are also set up to react to their environment. All characters follow an algorithm to determine their movement in the grid-like world. This algorithm prevents the characters from walking into spaces that are marked as blocked even if their key presses are telling them to walk into a blocked area. Characters react to the space they are currently located in too. They check their current position for entities and the type of entity at the position, if it is a power up, they react accordingly by modifying their attributes to reflect the power up effects.

The AI is a rather big portion of the game because, otherwise, the game wouldn't be much fun to play alone. The AI makes the game fun for those who don't have a friend readily available to play against. The biggest challenge of the AI was getting it to act in a manner that is mostly intelligent while keeping things simple to understand. One way in which this was achieved was to design the AI around a goal-based system. The AI will set a goal for itself and focus itself on achieving that goal before making another decision.

In designing the AI around goals, we were also able to achieve simplification in our coding which was to create separate functions which make a decision based on the desired result. The AI generates a random number and, based on probabilities and its current supply of bombs, decides what action it will take. The actions we have implemented include: attack, destroy, find, and avoid. Attack attempts to attack other players, destroy will attempt to destroy blocks on the map, find will seek out powerups, and avoid will try to avoid unsafe areas.

Each action function takes different aspects of the map into account. For example, attack looks for players that are reachable and targets the closest reachable character. If no characters can be reach, it forwards the action to destroy. Destroy takes into account the entities currently on the map, seeking out spots where a maximum number of blocks can be destroyed. Avoid makes decision based on mapping out unsafe areas and getting out of unsafe areas as quickly as possible. Finally, seek takes into account nearby entities to determine a spot to move to.

Entities, which have been vaguely touched in previous sections, are simple objects that exist within the map. Entities include destructible objects, powerups, bombs, explosion fire, etc. Each of these extends a base entity class which gives it a variety of functionality but also allows each entity to develop its identity. Destructibles block the player whereas powerups do not block the player. Based on a mask developed by each entity, the players and other entities can react uniquely for each situation.

Results:

The game plays pretty smoothly as intended. Each player/enemy starts in a corner of the map and begins with a bomb supply of one, as well as a blast radius for their bombs of two grid spaces. Players who touch flames created by bomb explosions will die, and these explosions will also destroy destructible blocks, will destroy power ups they touch, and will also chain detonate any other bomb within range.

The AI for the enemies is a bit iffy at times in the game but acts as a pretty good challenge for most players. Considering how none of us have taken an Artificial Intelligence course, it works rather well. The AI will randomly choose an objective based on our predetermined probabilities for each action and map out a path to complete it. The result of this was that it acts rather intelligently most of the time but under some conditions it will commit suicide by making a bad decision. However, the AI does have the ability to recover from a bad decision, if the AI spends too much time not moving when it thinks it should be, it is given a chance to reroute itself and possibly make a better decision.

The music runs in the background and loops as the game progresses. However because we do not have a proper sound handler we can only play one sound at a time, and we chose background music.

The player controlled character may move based on the WASD keys, drop bombs with "C" and we added a function to drop destructible blocks with "X" for debugging purposes and because it's fun. Movement is fluid, though bomb dropping sometimes is read as an input twice when on the move. Whenever a player is the last one standing, the game will reset and the last surviving player will gain one point in their score. Should the rare case where all remaining players die within the same frame happen, the game is also designed to trigger a draw where no player gain a point and instead, the game is reset.

Conclusion:

It's actually pretty fun to mess around with a game you've created. Creating an AI from scratch is really amusing in it's own right, as you start it off with simply trying to make it move based on a simple instruction (which for us, started off with a random probability based method that caused it to spaz out like it had Parkinson's). Run-time errors in general are really amusing too, there was a time where we forgot to implement a check for the keyboard input to be down for bomb drops, and the result was creating a huge chain of bombs wherever you walked, sort of like a deadlier version of snake. Overall, we are proud of the result of this project and believe that there is much left that could be improved. We could stray the design of the original game and implement our own fun power-ups, characters, and levels to create something fun and new.