Christina Fontana

Srinidhi Venkatesan Kalavai

David Kirk

Jillian Yong

EC327 Android Project Documentation - I Smite You

Brought to you by the Wizards of PHO307

**The Game:**

“I Smite You” is a tile-switching turn-based game where a user controls a wizard that defends against orcs. The orcs are trying to eat the wizard, so the object of the game is to stay alive by defeating all the orcs. Orcs are defeated by luring them into obstacles. These obstacles include walls, mines, bombs, and holes. There are 8 levels for the user to beat, and levels will increase in difficulty as the user advances. Each level is won once all the orcs end up stuck behind a wall, stuck in a hole, or blown up by a bomb. The user can switch tiles that contain orcs, bombs, or mines. Tiles with walls, holes, or the wizard cannot be switched.

The game features four different types of orcs of varying intellect: Basic, Smart, Brute, and Wary. Basic orcs have poor pathfinding and get stuck behind walls easily. The player only needs to place a wall in front of a basic orc to defeat it. Smart orcs have smart pathfinding and can navigate around walls. Brute orcs have poor pathfinding like the Basic orcs, however, they can destroy walls. Wary orcs have smart pathfinding and can avoid walls, holes, and bombs, but not covered holes.

Walls block the movement of Basic orcs, and slow the movement of Smart and Wary orcs. Holes can either be uncovered and visible to wary orcs or covered and invisible to all orcs. Uncovered holes can destroy Basic, Smart, and Brute orcs, but can only slow down Wary orcs. Covered holes are invisible and can destroy all orcs, but once an orc falls in, the hole turns into an uncovered hole and can be discovered by Wary orcs. Small black bombs destroy destroy only the orc standing on its tile. Once the bomb is used once, it vanishes. Large red bombs destroy all orcs in its own and neighboring tiles, and like the small bombs, can only be used once. Two orcs cannot take up the same space and will treat existing orcs as walls.

**Front End:**

The start screen will show the title of the app, a start button, and a level select button. The level select button allows the user to choose a level to solve. Otherwise, the start button will take the user through a tutorial explaining the elements of the game, and then immediately take the user to level 1. The introduction/tutorial shows images of the orc, wizard, and wall and explains their purposes. It explains how the game is played (by switching tiles), how you win (by blocking/destroying all the orcs), and how you lose (by getting eaten by the orcs). As the player taps on squares to switch, that square is highlighted in white to confirm that the tile is selected. The second tile the user taps is the tile that will switch with the first tile. Once the player switches two tiles, the player’s turn is over and the orcs all move one space closer to the wizard. If a Basic orc gets caught in a wall, it will stop moving. If any orc falls in a hole, it will be taken out of play. If a Brute orc smashes a wall, the wall is taken out of play. The game is played on a 9x9 grid of squares and varying numbers of orcs and obstacles.

**Back End:**

CartPoint Class:

This class creates locations that almost all objects will use to determine where to move and what tiles to switch. It has two public member variables (x and y) and two constructors. The default constructors initialize x and y to (0, 0) respectively, while the second constructor takes two double arguments and sets x to the first argument and y to the second argument. The class also includes public member functions to allow CartPoints to be added, subtracted, multiplied, divided, and compared. There is also a static function, cartDistance, which calculates the distance between two CartPoints and returns it as a double. This class is based on the similarly-named one from PA3.

GameObject Class:

This class creates objects that will be used in the game. It is the base class for all of the obstacles, orcs, and wizards. The protected member variables are a displayCode to identify what kind of object it is, id to identify the object if there are multiple of the same object, state to identify the state the object is in, and location, to identify where the object is located on the game board. The GameObject class does not have a default constructor, but its simplest constructor has a character argument that the display code is set to. This constructor also initializes id to 1, state to “s”, and location to (0, 0). The second constructor has 3 arguments: a CartPoint, an integer, and a character. The location variable is set to the CartPoint argument, id is set to the integer argument, and displayCode is set to the character argument. The state is initialized to “s.” The class has getter and setter functions to get and set the member variables, and it also has an abstract update function to allow polymorphic use of that function on GameObject’s children, and a toString function for printing. There is also a simple swap function is included in the GameObject class which allows the object to swap itself with another GameObject.

Wizard Class:

The Wizard class is derived from GameObject and contains all the functions that allow the wizard to move objects. The wizard can move orcs, bombs, and holes, and if the wizard dies, the game is over. The first constructor takes a Model argument and sets the state to “a” for alive, the location to (0,0), and the displayCode to “P” for Person. The second constructor takes a CartPoint argument and a Model argument, and sets the same variables as the first constructor, but sets the location of the wizard to the CartPoint argument.

The Wizard class also includes member functions magicSwap and update. The public magicSwap function is what allows the wizard to switch two tiles to defend against the orcs. It prevents the wizard from switching tiles that have walls, itself, or holes. Since the wizard does not move, the update function checks to make sure that the wizard is still alive, and if it is not, the function returns true and changes the wizard’s displayCode to indicate that it has died.

.AStarPathfinding Package:

This package includes a series of classes that allow the smart and wary orcs to pathfind their way to the wizard, and was written by Kevin Glass and found on his website here: <http://www.cokeandcode.com/main/tutorials/path-finding/>. It contains a version of the A\* pathfinding algorithm that was well-suited to this game, so the game was adapted to use it. Generally, orcs will move in straight lines towards their targets, but the smarter orcs will avoid specific obstacles while the more basic orcs will collide with any obstacles in their way, with different results depending on what the obstacle is. The *Path class* contains functions that tell the object where to step and how to change direction if it is avoiding an obstacle. Smart and wary orcs are the only ones that use the AStarPathFinding package.

Wall Class:

The Wall class is derived from GameObject and it contains the number of walls in the level and a function called by a brute orc that will smash the wall. The constructor sets the location of the wall to CartPoint argument and increments the number of “intact” walls (not broken by brute orcs). The smash function sets the wall’s state to “d” for “destroyed” and moves the wall to a tile outside of the board to take it out of play.

Hole Class:

The Hole Class is derived from GameObject and it contains the number of holes in the level, and whether those holes are covered (state = ‘c’) or uncovered (state = ‘u’). The first constructor has a CartPoint argument that sets the hole’s location to the CartPoint, and the second constructor has a boolean isCovered argument that determines if the hole that is constructed is covered or uncovered. The hole’s update function detects if an orc has fallen in so if it is a covered hole, it switches to uncovered.

Bomb Class:

The Bomb class is derived from GameObject and it contains the number of bombs in the level. Bombs can either be large red bombs that blow up orcs within a 1 tile radius (state = “R”) or small black bombs that blow up orcs that land on that tile (state = r). The object senses if there are orcs in the vicinity, and shows the explosion animation if they come close enough. Bombs do not affect Brute orcs, and red bombs can kill the wizard if the wizard is within the explosion radius. Once a bomb is used, it is taken out of play and can no longer be used until after the level resets.

GameWorld Class:

The GameWorld Class creates the back end view for the game through an array matrix. In this matrix, each GameObject is represented as a single character, so each unique state an object can be in has a character representation here, including “effects” like explosions—the only exceptions are mines, which are supposed to be invisible to all orcs. This class is used primarily by the smart and wary orcs, since its implementation was required to get the A\* pathfinding code to work. The default constructor creates the 9x9 matrix for the backend and the class itself has functions to plot, clear, and block movements of orcs.

Model Class:

The Model class holds all the GameObjects for each level as well as the GameWorld used for pathfinding. It loads the levels from text files, and generates new GameObjects at a given location and gives them a reference to itself to allow them to “see” their surroundings. The member variable gameState determines whether the user has won the level, lost the level, or is in progress of playing the level. It contains the information for creating the 9x9 board and prints all the objects as per the text files containing the levels. The class checks if the level is viable, and updates all the objects in play.

Orc Classes:

The base class orcs called the Orc Class is derived from GameObject, and contains the information for the basic orc. It has functions that allow the orc to move, stop , die, and use pathfinding to find its way to the wizard. The OrcBrute, OrcSmart, and OrcWary classes are all derived from the base Orc class, and behave in a manner explained in the game explanation.