**Project Proposal: Lemmings REVENGE!**

Author: Sean Greene

I. **Objective**

This program is a game written in Java with a SWING GUI interface. The game is a take on the original game “Lemmings” published by Psygnosis for the Amiga system in 1991. The objective of the game is to advance a set number of characters, defined by each level, from the level entrance to the level exit. The characters are autonomous, following a set of programmed behaviors (moving from right to left horizontally, changing direction on collision with an obstacle). The player can optionally assign a set of “skills” to each character, which enable that character to alter the level’s environment or effect the behavior of other characters. The user must strategize the selection, assignment, and timing of these skills to overcome the various level obstacles and clear a path for the remaining characters to advance to the exit and fulfill the level requirement.

II. **Motivation**

This program is a demonstration of using Java and Swing for basic game design. The program will include a number of Swing GUI components to create the game window, menus, and dialogs for instructions, hints, and level selection. It will push the limits of Swing components by creating custom renderings, event handling, collision detection, and autonomous character animation. The user interacts with the autonomous characters through mouse listener events.

III. **User Interface**

User will interact with the GUI through mouse clicks. The user clicks on one of the characters, which displays that character’s current stats in a side panel. Clicking on a skill after selecting the character assigns that skill to the character. A count displaying the number of characters successfully through the exit vs characters remaining vs characters required for the level is displayed on a panel in the top right of the frame.

IV. **Basic Design and Methods**

The game will follow a Model-View-Controller architecture.

Models include:

1. Character classes defining the characters
2. Skills classes and Skills interface for defining skills
3. Level class which includes the current score, countdown timer, Obstacles

View includes:

1. GamePanel class for displaying the level, characters, environment
2. Menu class for displaying the score, timers, and menus.

Controller includes:

1. LevelController class:
   1. Receives mouseClick events and updates the Character models.
   2. Contains a Timer object which calls the LevelController.actionPerformed() that updates the game state.

V. **Classes and high-level methods**

Level class: Model containing the game state.

* ArrayList<Character>: For referencing the characters on screen.
* Int[][] map: a 2d array representing the level. Assigning 1 = isGround, 0 = air, 3 is a solid object, which will invoke an onCollision().
* Game clock, score, progress toward goal.

LevelController class: Updates the GamePanel views and the models (Character model, Level model). Constructor would need to include the Level model and the GamePanel view. Includes a Timer object for calling actionPerformed and refreshing view every x number of frames.

* addMouseListener(MouseAdapter): for detecting which character in array is clicked and updating that character. Displays the selected Character’s stats on the scoreboard panel.
* actionPerformed(): invoked by a Timer object, this method will update the game with new character positions and changes to the level’s environment. Calls repaint();
* updateGame(): updates character positions and changes to obstacles, scoreboard.
* isGround(int char\_y\_pos, int char\_x\_pos): Boolean method for determining if a character is on the ground. Returns true if the character’s Rectangle x, y coordinates cross ground.
* onCollision(int char\_y\_pos, int char\_x\_pos): Boolean. Returns true if a character’s Rectangle x, y coordinates cross an obstacle.

GamePanel class: The main JPanel that renders the level, environment, and characters. Includes an ActionListener for detecting Timer events, and a mouseListener for determining which character was selected.

* enum of SKILLS
* createLayout(): handles the graphics components.
* paintComponent(): renders the characters and obstacles on the game panel.

Character class: An abstract class that defines a basic Character. It oversees tracking the position of a character on the screen and setting the character’s direction.

* update(): updates the character’s position. It will also include a loop for a very basic animation of the character.
* draw(Graphics g): draws the basic character. Will include a g.drawImage for the character images.
* toggle(): Toggles a Boolean instance variable for determining direction of character movement.
* getBounds(): draws a rectangle around the character, to be used with mouse events.
* onCollision(): A method for determining collision events with obstacles.
* abstract assignSkill(Skill skill): Method assigns the selected skill to the clicked character.

Obstacles class: Abstract class for defining obstacles. Calls the characters onCollision(String obstacle) method when it detects a collision.

Lemming class: Extends the Character class. It implements the methods in the character class.

* Overrides update for lemming images

Interface Skill: single method interface for defining a skill.

* useSkill(Character c)

Builder class: implements Skill.

* Overrides useSkill(Character c): codes actions for building a bridge object. Will detect when this character encounters a gap and will create a bridge of 1 unit long.

Digger class: implements Skill.

* Overrides useSkill(Character c): codes actions for removing a single brick unit from the board

Blaster class: implements Skill.

* Overrides useSkill(Character c): codes actions for destroying obstacles. Will detect when this character encounters an obstacle and remove it from the board.

Blocker class: implements Skill.

* Overrides useSkill(Character c): codes actions for blocking other players, effectively making this character an Obstacle.