MAZE - Multiple Adventure Zone Environments
("Get lost in our language" - MAZE developers)
Project Proposal

Alexander Brown (aab2212), Alexander Freemantle (asf2161),
Michelle Navarro (mn2614), Lindsay Schiminske (ls3245)
September 28th, 2016

Describe the language that you plan to implement. Explain what sorts of programs are meant to be written in your language Explain the parts of your language and what they do Include the source code for an interesting program in your language 2–4 pages

Language description:

MAZE allows game designers to create text-based games. The core building blocks of MAZE are cells. Cell style map design allows for game designers to enforce movement rules. Players can move around a string of cells like a board game or players can move in a more free-form style jumping from cell to cell. Game ending conditions can be enforced by the players movement or other factors like health, score or a time limit. Depending on the type of game the developer creates, each cell is able to contain an item.

What sort of programs are meant to be written?

Everything from a board game to an adventure style game can be written in MAZE. The types include player, cell, item and objective. Our language will include predefined cell style types (ie. Board Style) but it will also allow for designers to spec their own cell layout.

Parts of the language:
//this is a comment
Primitives:

Type	Definition	Example
int	signed integers	-1, 0, 5
boolean	boolean values	true, false
string	set of characters	"MAZE is awesome"
char	single character	'm' 'a' 'z' 'e'

Keywords:

Keyword	Description
If, else	Control flow statements
while	Loop statements
extends	for subclass definitions

Operators:

Operator	Description
+, -	Addition,Subtraction
*, /, %	Multiplication, Division, Mod
== , !=	Equivalence
>, <, >=, <=	Inequality Operators
&&, ||, !	Logical AND, OR, NOT

Built-in Classes:

Class	Description	Fields	Interface
World	There can only be one world per game. Holds lists of players, items, cells.	**size** *int* The number of cells in the world **playerList** *dict* The names of the players in the world. The key is the player name **itemList** *dict* Items in the world. The key is the item name **cellList** *dict* Cells with their names as keys	**listen()** Listens for user actions & changes game state appro **isEmpty()** Returns true if all cells are empty **isEmpty(name)** Returns true if a specific cell is empty **isFull()** Returns true if all cells contain an item

		timeLimit *int* in minutes, (if 0 the time limit is not enforced) **actionList** *dict* Actions available for players. Default actions are the built-in directions (south, north etc.)	**insert(Cell)** Add a cell to the world **remove(Cell)** Removes cell **winGame()** Triggers the end of the game, displays win message **loseGame()** Triggers the end of the game, displays lose message

Class	Description	Fields	Interface
Cell	An object that holds items	**name** *string* The name of the cell **itemList** *Item* The contents of the cell **south**, **north**, **west**, **east**, **sw**, **se**, **nw**, **ne** Links between cells (these are bi-directional)	**insertItem(Item)** Insert Item into cell

Class	Description	Fields	Interface
Item	Any object inside of of a cell or belonging to a player	**name** *string* The name of the item	**toString()** Returns the name of the item

Class	Description	Fields	Interface
Player	An actor in the game	**name** *string* The name of the player **currCell** *Cell* The player's current cell **hasWon** *boolean* True if the player has won **ItemList** List of items that the player has	**toString()** Returns the name of the player **giveItem()** Add item to the player's itemList **hasWon()** returns true if the player wins

Class	Description	Fields	Interface
Action	Choice the player is able to make	**name** The name of the action	**toString()** Returns the name of the action

```
1 Sample Code:
 2 //Simple maze game
 3
 4 Class mazePlayer extends Player {
 5     boolean hasWon(Cell winCell) {
 6       return currCell == winCell;
 7    }
 8 }
 9
10 createPlayers {
11     mazePlayer = mazePlayer(name: “Mandella”);
12     mazePlayer.hasWon(maze.centralPark);
13 }
14
```

```
createMaze {
    maze = World();
    maze.actionList = [“north”, “south”, “east”, “west”]
    fuzzyRoom = Cell(msg:”Welcome to the fuzzy room. Which way to Columbia?”);
    lowSteps = Cell(msg:“Good work, you have made it to the steps. \
       This game is a walk in the park”);
    centralPark = Cell(msg:”Congratulations! You made it to the park. Enjoy the sun.”);
    fieryPit = Cell(msg:“Whoops, you chose the wrong path and you died. RIP”);

    //define the starting point
    maze.startRoom = fuzzyRoom;
    maze.winCell = centralPark;
    maze.loseCell = fieryPit;

    //describe path
    fuzzyRoom.east(“lowsteps”);
    lowsteps.south(“centralPark”);
}


play {
    //user must end the game
    createMaze();
    createPlayers();

    //start game
    currCell = startCell;

    while (mazePlayer.currCell != centralPark && mazePlayer.currCell != fieryPit){
        print(currCell.msg);
        listen();
    }

if(currCell == centralPark){
    winGame();
}
else{
    loseGame();
}

```