Maze Game: A Series of Homework Assignments
Software Design Document

Keith Garfield
Version: Draft 2
Date Oct 16, 2022
CS 225, Fall 2022

Embry-Riddle Aeronautical University
Daytona Beach campus
1 Aerospace Boulevard
Daytona Beach, FL 32114

INTRODUCTION:

The assignments summarized in this document will result in a small project called the Maze Game, in which autonomous agents attempt to solve a randomly generated maze puzzle. This document provides a complete description of the Maze Game, followed by a description of a software solution to be implemented by the students. By completing this series of assignments, students experience a complete project development exercise in a guided format. In addition, students are provided with examples of project deliverable documents: the Software Design Document (this document), the Agile Tracking Sheet, and the Requirements and Test Document. These examples provide guidance to students on the content and style of project deliverables.

The Maze Game will be implemented by students through the assignments summarized in Table 1. Terminology used in Table 1 is explained more fully in the Problem Description and Problem Solution sections of this document. Complete descriptions of the requirements and grading rubrics used for each assignment are provided separately for each assignment. Due dates are provided on the course Canvas assignment tool.

Table 1: Assignment Summary

HW ID	Description	Deliverables
HW3	Create a maze using only generic tiles	MGMain.java GameController.java GenericTile.java
HW4	Create an agent that moves within the maze by choosing directions randomly.	HW3 plus - GenericAgent.java
HW5	Loading a maze from a file to start the game, and recording the agent's maze traversal to a file.	HW4 plus – maze_01.dat result.dat
HW6	Extending the GenericTile class. Create a maze composed only of Static, Rotating, Glass Wall, and Solid tiles.	HW5 plus – StaticTile.java RotatingTile.java GlassWallTile.java SolidTile.java
HW7	Extending the GenericAgent class. Create Greedy and Left Wall agents to traverse the maze.	HW6 plus – GreedyAgents.java LeftWallAgent.java

PROBLEM DESCRIPTION:

This Maze Game constructs a maze using a 2-dimensional (2D) array of spaces referred to as tiles (Figure 1). The maze is composed of an n-by-n square array of tiles, where n is an odd integer greater than 0. Tile locations are designated in column, row order (col, row). A single tile is marked as the Start location, and a single tile is marked as the Goal location. The purpose of the game is for an autonomous agent to traverse the maze from the Start location to the Goal location in the fewest number of moves. The Start tile is always the tile on the center row of the

western (left hand) edge of the maze. The Goal tile is always the tile on the center row on the eastern edge (right hand side) of the maze.

The agent wins the game if it reached the Goal tile. The agent loses the game if it does not reach the Goal tile in 50 moves. Every attempt at a move counts as a turn, whether the move was successful or not.

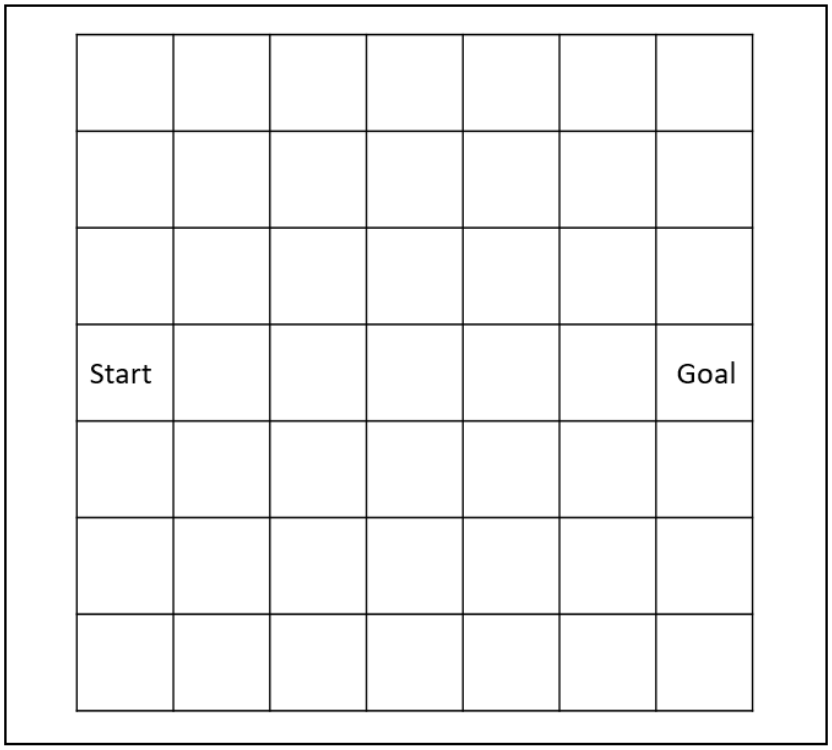

Figure 1: Maze Game 2D Tile Array

Each tile is square, with horizontal and vertical edges, labelled as if on a map with the north (N), east (E), south (S), and west (W) edges (Figure 2). Each edge contains a single exit, which may be locked or unlocked. Agents traversing the maze may only move from one tile to another through an unlocked door. Note that a single door may be unlocked inside one tile, and locked inside the neighboring tile, allowing only one way movement through the unlocked side of the door. The configuration of locked and unlocked doors is determined randomly at the start of the game, with the likelihood of any door being locked to be determined later as part of the game development. The agent is not allowed to move outside of the maze.

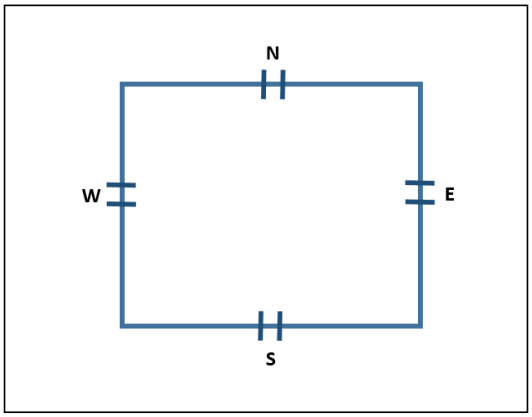

Figure 2: Tile Configuration

In addition to doors, each tile may perform up to three actions (Table 2): an *Enter Action* that is triggered when the agent enters the tile, an *Exit Action* that is triggered when an agent leaves the tile, and a *Special Action* that may occur at any time during the game (the likelihood of a special action occurring will be determined during game development). Generic tiles are not intended to be used in the game, but actions are presented for completeness (Table 2).

Table 2: Tile Actions

Tile Type	Enter Action	Exit Action	Special Action
Generic	None.	None.	None.
Static	None.	None.	None.
Rotating	None.	Tile rotates 90 degrees clockwise.	Tile rotates 90 degrees counterclockwise.
Glass Wall	Agent is provided with the door configurations for all surrounding tiles.	None.	None.
Solid	Entry not allowed.	None.	None.
Note: Students may choose to create additional type of tiles with new actions.			

The maze is traversed by a single autonomous agent attempting to move from the Start tile to the Goal tile. The agent may move only to an adjacent tile through an unlocked door. There are two types of agents used in the game (Table 3). Generic agents are not intended for use in the game, but behavior is presented for completeness.

Table 3: Agent Behavior

Agent Type	Agent Behavior
Generic	Each move is decided randomly, with an equal likelihood of moving through any unlocked door in the current tile.
Greedy	The agent always selects an exit that moves it closer to the Goal tile. If there is no exit available that moves closer to the goal tile, the agent will select any exit.
Left Wall	The agent follows the maze solving rule of always following the left hand wall. The agent will always choose to take unlocked doors starting from the left hand side (relative to how the tile was entered) and moving clockwise. The agent will not backtrack to the tile it immediately exited if another option is available. Example: Agent enters a tile having all four doors unlocked. If entering from the south, the agent will exit using the west door. If entering from the north, the agent will exit using the east door. In both cases, the agent chose the leftmost option.
Note: Students may choose to create additional types of agents with new behaviors, or to create an agent allowing a human player to play.	

PROBLEM SOLUTION:

Note: This section provides an initial design, which may be modified as the software is developed.

This section provides a description of the software implementation of the Maze Game. An overview of the software design is provided in a Unified Modeling Language (UML) class diagram (Figure 3), followed by a detailed description of the purpose and methods available to each class. This discussion does not explicitly discuss “setters and getters” for the class attributes, which are assumed to exist as needed.

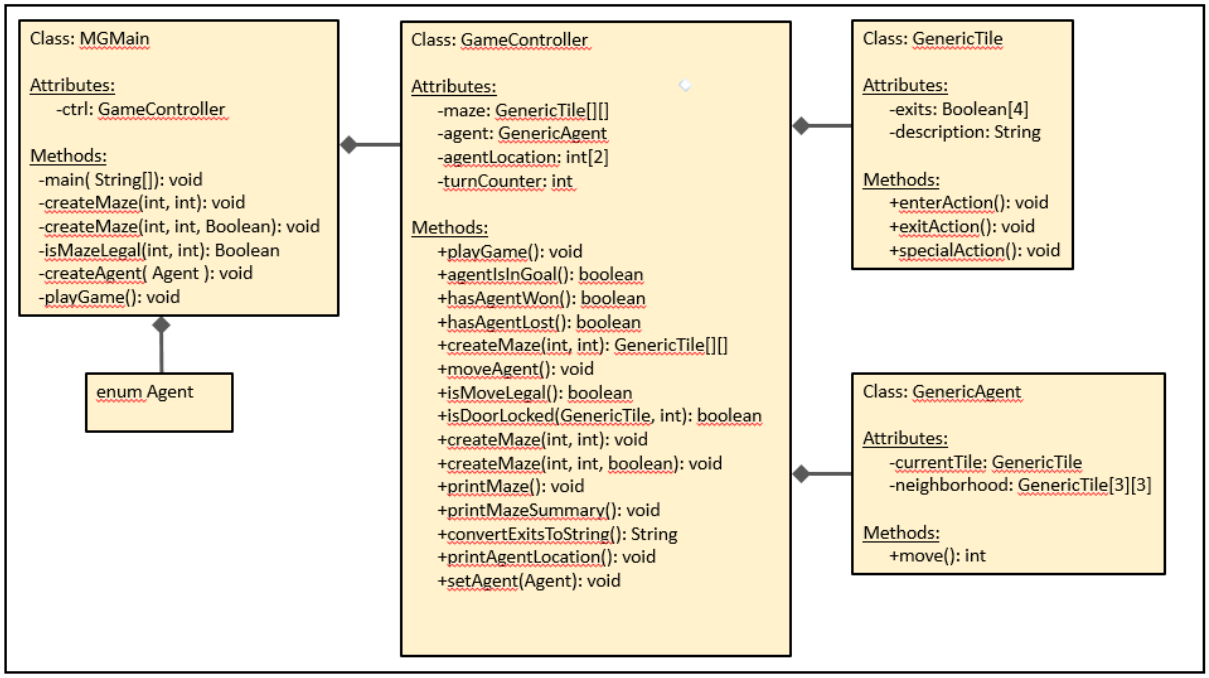

Figure 3: Maze Game UML Class Diagram

The overall flow of action and information for the game is listed below.

1. The MGMain class creates an instance of the GameController class and requests the GameController to initiate and run the game.
2. The GameController class creates all tiles in the array.
3. The GameController class creates an autonomous agent and places it in the Start tile.
   3.1. The GameController places the agent in a tile by passing the tile handle to the agent using the agent's setCurrentTile() method.
4. The GameController iteratively requests a move from the agent, the agent returns a request to move in a specific direction (N, E, S, or W), and the GameController passes the agent the handle of the tile being moved into.
   4.1. If an illegal move is requested, the GameController requests the agent to try again. The agent class keeps track of which directions have been attempted for the current tile.
5. The GameController ends the game when the agent moves into the Goal tile. In some versions, the GameController may limit the number of turns allowed to the agent.

Details of each class are provided below.

MGMAIN CLASS –

This class provides the initial starting point to run the Maze Game. This class is responsible for the following actions:

1. Create an instance of the GameController class.
2. Specify whether the maze will be read from a file or created randomly.
   2.1. File name must be specified if reading from a file.
   2.2. Specify maze size if creating a random maze (if reading from a file, size is in the file).

3. Specify what type of agent is traversing the maze.
4. Command the GameController to execute the game.

GAMECONTROLLER CLASS –

This class initializes the game by creating the array of tiles forming the array either through file input or random generation, creating and placing the agent in the Start tile, and iteratively requesting a move from the agent until the agent reaches the Goal tile. This class is responsible for the following actions:

1. Create the maze either randomly, or with all doors locked/unlocked, or read from a file using the GenericTile class or children of that class.
2. Create an instance of the GenericAgent class (or a child class), assigned to the player attribute, and initialize in the Start tile.
3. Track the agent location using the (col, row) tile coordinates stored in the playerLocation attribute.
4. Request a move from the player class.
5. Move the player agent as requested and provide the agent with the handle of the tile moved into using the agent's setCurrentTile() method.
   5.1. If the agent moved into a GlassTile, provide the agent with the current tile plus the eight surrounding tiles by setting the agent's neighborhood attribute.
6. End the game if the agent enters the Goal tile (in some versions, the game may be ended by exceeding an allowable turn count).

GENERIC TILE CLASS –

This class provides the description of an individual tile. Each tile has four doors which provide exit if unlocked. Locked and unlocked status is determined randomly. Each tile has an entry action, exit action, and special action. All of these actions are null for the GenericTile class. This class will be extended to create the specific variations of tiles in the maze. The tile classes are responsible for the following actions:

1. Performing the enterAction actions.
2. Performing the exitAction actions.
3. Perforing the specialAction actions.

GENERIC AGENT CLASS –

This class traverses the maze through a series of moves provided in response to the GameController's request for a move. The moves made by the GenericAgent class are determined randomly from the available unlocked exits in a tile. This class will be extended to provide more sophisticated maze solutions. Agent classes are responsible for the following actions:

1. Determine a move (in response to a request from the GameController)
2. Track attempted moves for the current tile (not performed in the GenericTile class).

REFERENCES:

No references are needed for this document.

APPENDICES:

No appendices are needed for this document.