### Technical Design Document Template

**1.0 Revision History**

|  |  |
| --- | --- |
| Version | Description |
| 1.0 | Initial document |
| 1.1 | Fill out Development Environment |
| 1.2 | Fill out Game Overview |
| 1.3 | Update Game Flow |
| 1.4 | Fill out Mechanics |
| 1.5 | Add Bomberman example |
| 1.6 | Add State Machine flowchart |
| 1.7 | Fill out interface |
| 1.8 | Add flee behaviour |
| 1.9 | Remove Flee Behaviour |

**2.0 Development Environment**

**2.1 Game Engine**

Unity Free Edition

**2.2 IDE**

Visual Studio 2019

**2.3 Source Control procedures**

GitHub

**2.4 Third Party Libraries**

**2.5 Other Software**

MSPaint

**3.0 Game Overview**

* 1. **Technical Goals**
* Working A\* Algorithm
* User Input
* AI Agents

**3.2 Game Objects and Logic**

* Maze
* Grid Object
* Moving Agent
* Change Spawnable Button
* Modify Maze Input Buttons

**3.3 Game Flow**

The game will immediately start with the player being able to move around by the user left clicking in the target position.

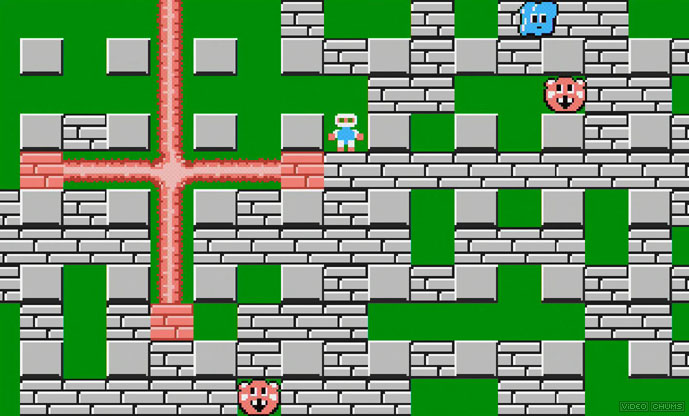
The user will be able to spawn in whatever they want such as walls, enemy agents and maybe more.

**4.0 Mechanics**

* The player can be moved around by left clicking on a white tile.
* The player will avoid walls.
* The user can spawn in objects
* The user can modify objects.

**5.0 Graphics**

Basic top down 2D style. Characters will have simple animations. Graphics can be compared to bomberman.



**Figure 1 – Bomberman example**

**6.0 Artificial Intelligence**

Agents will use a state machine with 3 states (IDLE, PATROL and CHASE)

* On Idle nothing will really happen with the agent other than potential animation change.
* On Patrol a random coordinate around the agent will be selected and the AI will move to it by using chase state.
* On chase the agent will travel to the target coordinates.

**Diagram

Description automatically generated**

**Figure 2 – Flowchart of Agent State Machine**

**7.0 Physics**

Unity was the chosen as it was the best choice for simple visualisation with as little code as possible. It also comes with its own coordinate/physics system which saves a lot of time.

One of the nice features is its Circle cast function. This sends out a circle ray cast that creates an array of all physics-based game objects in the circle. This makes implementing a sensor extremely simple; as all I will have to worry about is how often it is called.

**8.0 Items**

**9.0 Game Flow**

**9.1 ‘Mission’ / ‘Level’ structure**

Levels are set out however the player sees fit. This is because the player can spawn and remove walls wherever they want. They can also spawn enemies wherever and however they want.

**9.2 Objectives**

The player is in their own sandbox where they can test out the A\* algorithm by spawning enemies which attempt to hunt down the player.

Player can also create walls.

**10.0 Levels**

**11.0 Interface**

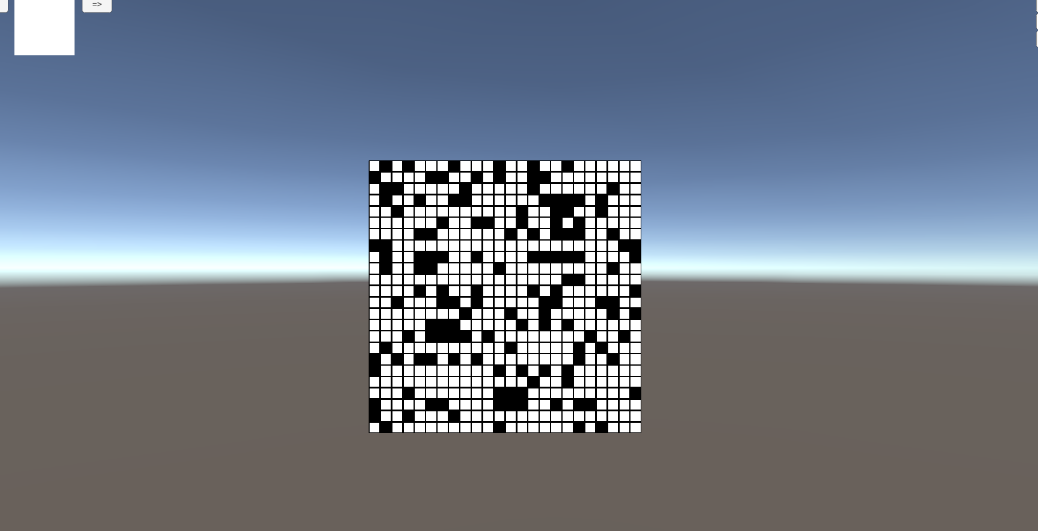
**11.1 Menu**

The only menu options present will be the users editing tools such as currently selected spawn able object.

The user can also pause the game using the escape key to see instructions on all the hotkeys. This includes spawning objects and drawing walls.

**11.2 Camera**

The camera will be not move at all. It will use perspective and will simply just be atop the maze rendering all the textures.

****

**Figure 3 – Camera view showing layout of the game (will take up whole screen)**

**11.3 Controls**

Mouse and Keyboard Is all that is required for controls. Left click for movement, right click to spawn/delete object.

**14.0 Asset List**

* Square Sprite – Used for everything (tile, agents, items)

**16.0 Technical Risks**

Need to make sure that spawning in a lot of enemy agents will not massively break game performance wise. This can be achieved by not calculating a path every frame, but instead once the agent gets the player within sight.