Triple Action – Growing & Resizing & Maze

Table of Contents

Triple Action – 3-Phazed Mobile Game 2

1. Core Loop && Progress 2
2. Code Structure – UML 3
3. Developer’s Diary 3

Initial Maze Approach 5

Implementation Resources: 7

Triple Action – 3-Phazed Mobile Game

1. Core Loop && Progress

Outline

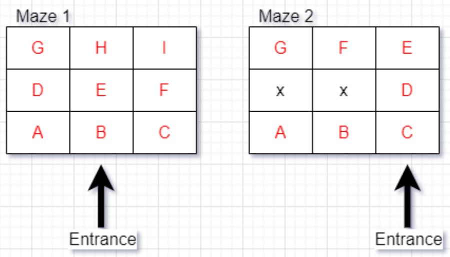
- Third-person camera
- growing + resizing + maze
- growing: collect other same color balls
- resizing: fit into obstacles
- maze: amaze!!! --- https://apps.apple.com/us/app/amaze/id1452526406
- win condition: complete maze
- fail condition: die or cannot complete maze
- Game progress should persist between sessions.
- 3D.
- UI should include basic elements such as Play, Retry, Next, Home buttons etc. for an uninterrupted gameflow.
- should not be any errors during the runtime.

Core-Loop Pseudocode && Detailed Outline

Calculating Minimum Required Balls to Complete Maze:

- ~~This is a tree problem: What is the shortest path to visit all nodes?~~
- ~~This problem can be solved by reducing it Travelling Salesmen Problem~~
- ~~Calculate the Minimum distance (cost) between every node.~~
- ~~Then create a complete graph with these values~~
- ~~Finally solve Travelling Salesmen Problem for this graph~~
- ~~Result cost will be the minimum required number of balls.~~

- ~~Appropriate way to solve this problem is using **Kruskal's Minimum Spanning Tree**~~



Graph 1

set of vertices V = {A,B,C,D,E,F,G,H,I}

set of edges E = {AB, AD, BC, BE, CF, DG, DE, EH, EF, IF, GH, HI}

Win UI:

- Top-middle -> Success Text
- Middle-middle -> Next Button
- Middle-bottom -> Home Button

Fail UI:

- Top-middle -> Fail Text
- Middle-middle -> Retry Button
- Middle-bottom -> Home Button

Main Menu UI:

- Top-middle -> Current level number label
- Middle-middle -> Tap to start label

In-Game UI:

- Top-left -> minimum needed balls to complete maze
- Top-right -> current number of balls
- Top-middle -> slider showing current position of player

Growing Phase:

- show tap to start
- go left-right with joystick
- grow by getting same colored balls
- shrink by getting other balls

Resizing Phase:

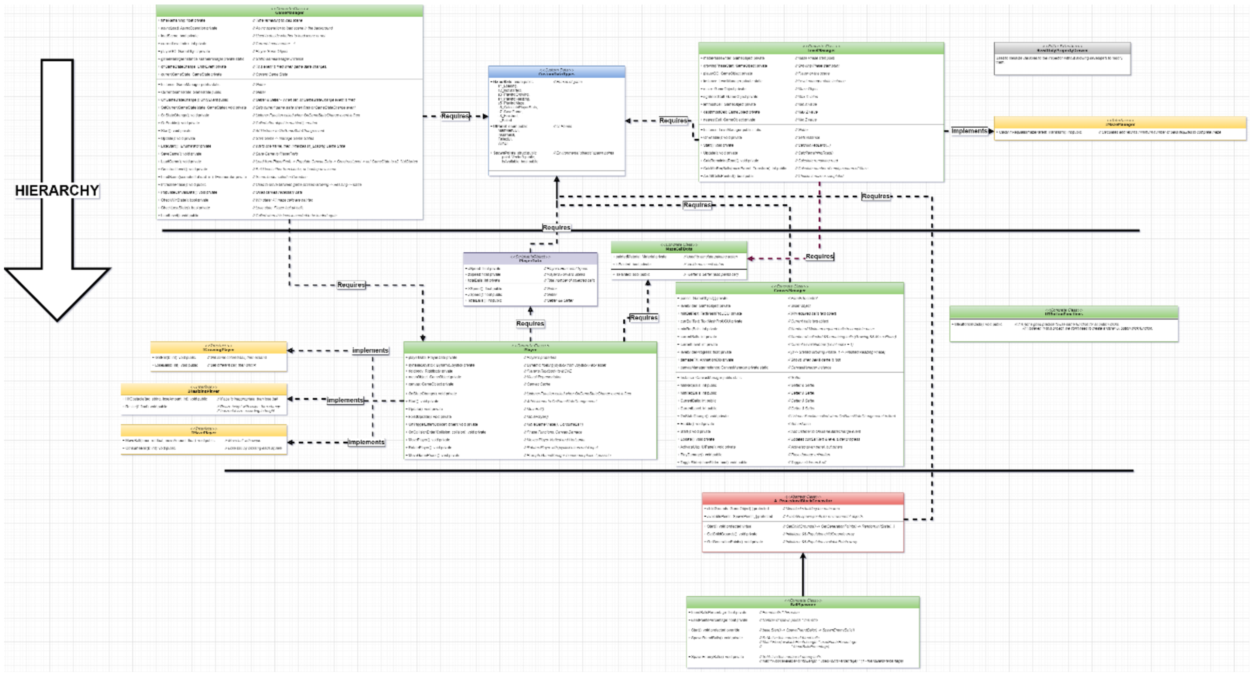
- Player cannot go left or right
- move your finger upward or downward to:
  - upward: to shrink down to slim vertical stick
  - downward: to expand to slim horizontal stick
  - Somewhere between: fat small box

Maze Phase:

- Crossing square consumes some ball
- Player should paint all empty squares without consuming all of himself/herself.
- Player still consumes itself if she/he goes on the painted square.

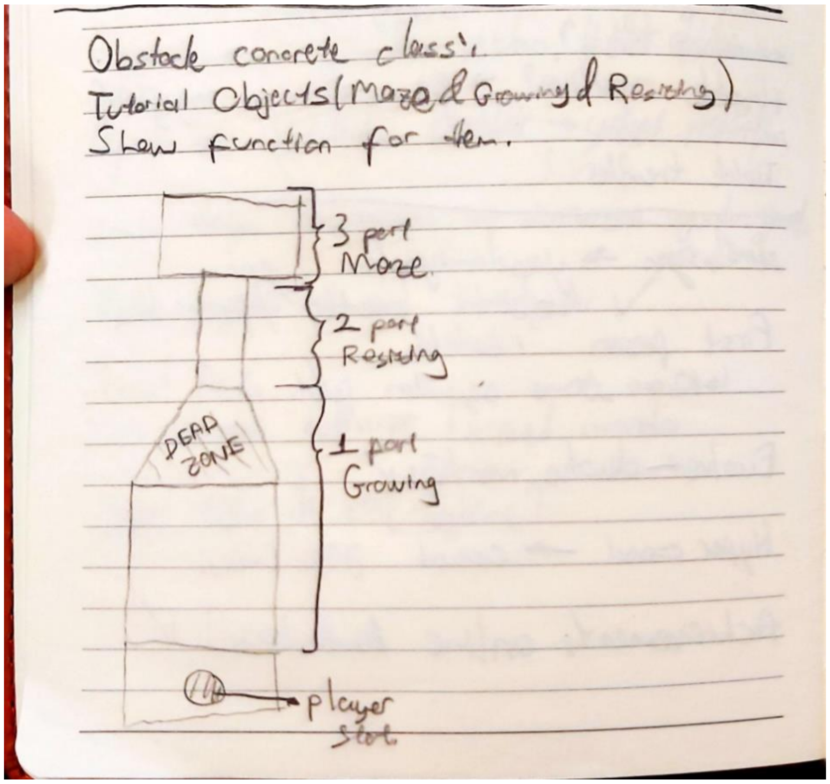
2. Code Structure – UML

Link to view easily: View on Web



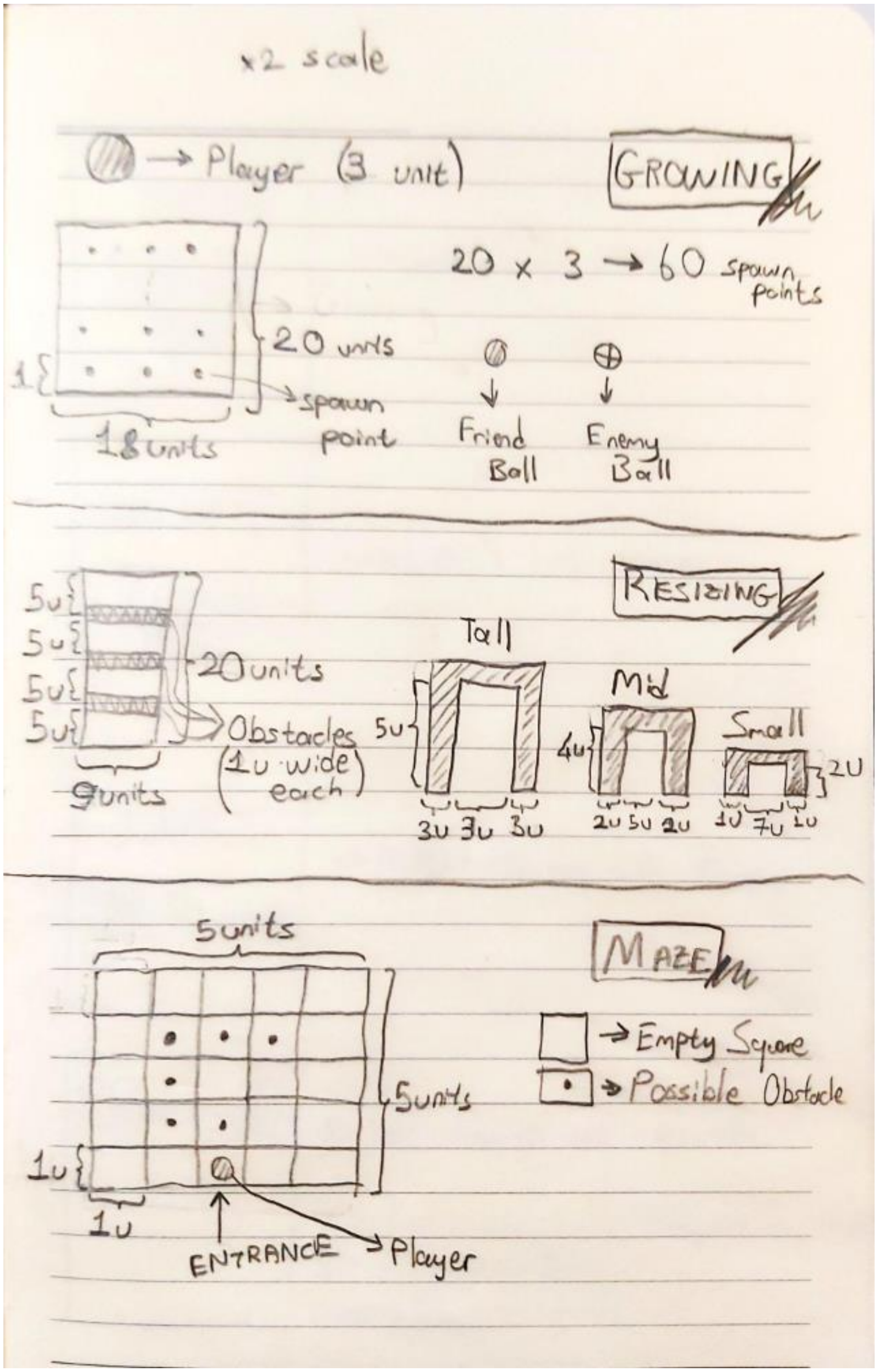
3. Developer's Diary

- First map design was different. Since I have a scale inconsistency between player and scene; I quitted this design:



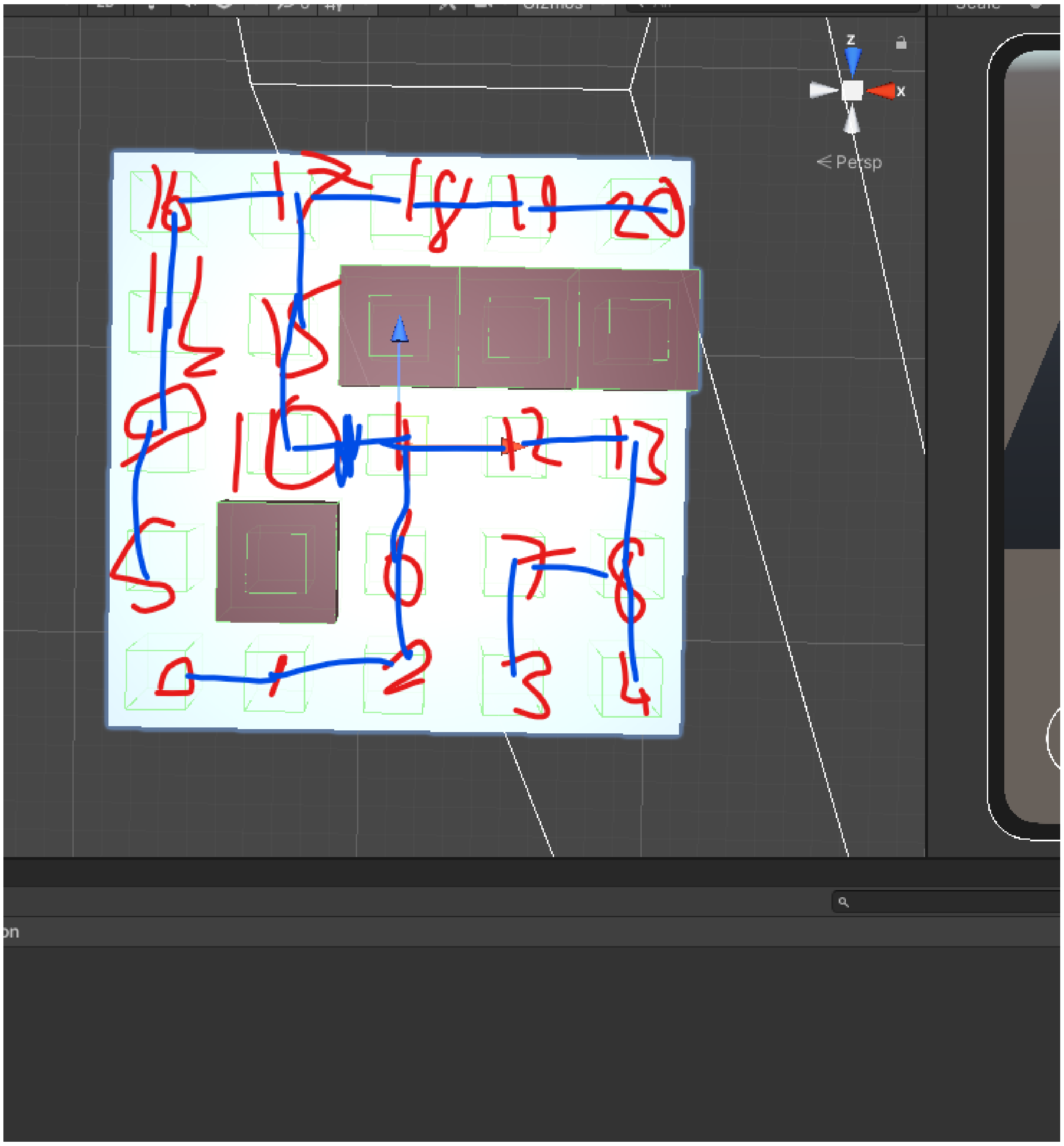
Obstacle concrete class',
Tutorial Objects (Maze & Growing & Resizing)
Slow function for them.

- Scale inconsistency also affected parts of whole level. Phases' sizes are different from initial design. Following data of Player's size is pointing wrong value.



Initial Maze Approach

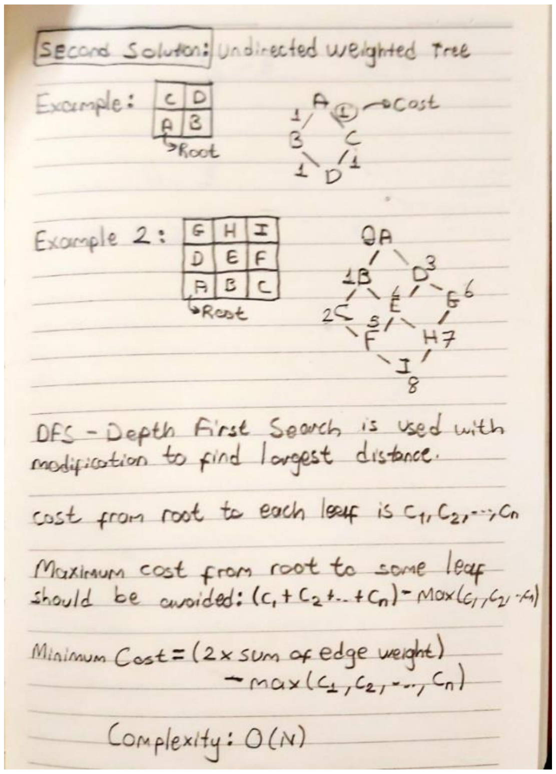
My initial approach for the Maze Phase was calculating minimum required balls to complete the maze.
Ex:



- To solve this, I tried following solutions and encountered following problems:

Undirected Weighted Tree – Minimum Spanning Tree in A Graph

- I managed to find an MST in a graph. Maze cells construct a graph having edges between them. But the problem is, it is calculating the minimum cost to cover all nodes of the tree. All connections are evaluated once: which means we don't traverse graph, rather than summing costs of edges of graph.



Second Solution: Undirected weighted Tree

Example:

C	D
A	B

Root

A
1
B
1
C
1
D
1
cost

Example 2:

G	H	I
D	E	F
A	B	C

Root

0A
1B
D3
2C
4E
G6
5F
H7
I
8

DFS - Depth First Search is used with modification to find longest distance.

cost from root to each leaf is $C_1, C_2, \ldots, C_n$

Maximum cost from root to some leaf should be avoided: $(C_1 + C_2 + \ldots + C_n) - \mathrm{Max}(C_1, C_2, \ldots C_n)$

Minimum Cost $= (2 \times \text{sum of edge weight}) - \max(C_1, C_2, \ldots, C_n)$

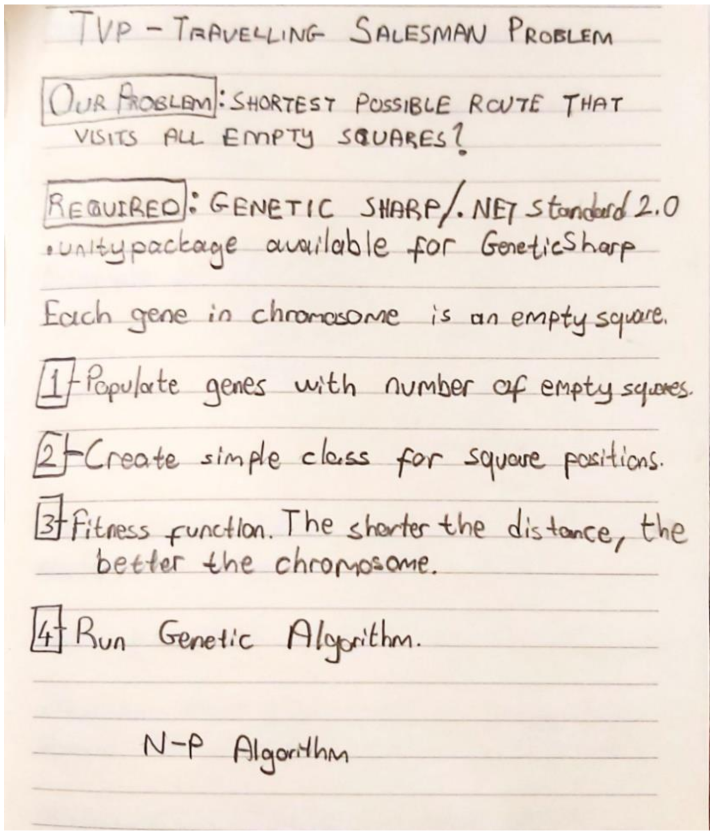
Complexity: $O(N)$

Travelling Salesman Problem – Pure Implementation

- This was my most hopeless try 😊 Since this is an NP-Hard problem, I couldn't manage to calculate the maze having 20+ cells.

Travelling Salesman Problem – Genetic Algorithm Approach

- NP-hard problems are situations in which there is no known algorithm to solve this problem in O(n) time. These problems need to be solved approximately with the most minimum error. One way to solve TSP is using Genetic Algorithm.
- I also managed to use this algorithm. However, the main problem is that it was calculating direct distance between two points in space. Which means that it considers every distance between two points (points are maze cells) a reachable edge. This leads another failure on my side 😊



TVP - TRAVELLING SALESMAN PROBLEM

OUR PROBLEM: SHORTEST POSSIBLE ROUTE THAT VISITS ALL EMPTY SQUARES?

REQUIRED: GENETIC SHARP/.NET Standard 2.0
.unitypackage available for GeneticSharp

Each gene in chromosome is an empty square.

1- Populate genes with number of empty squares.

2- Create simple class for square positions.

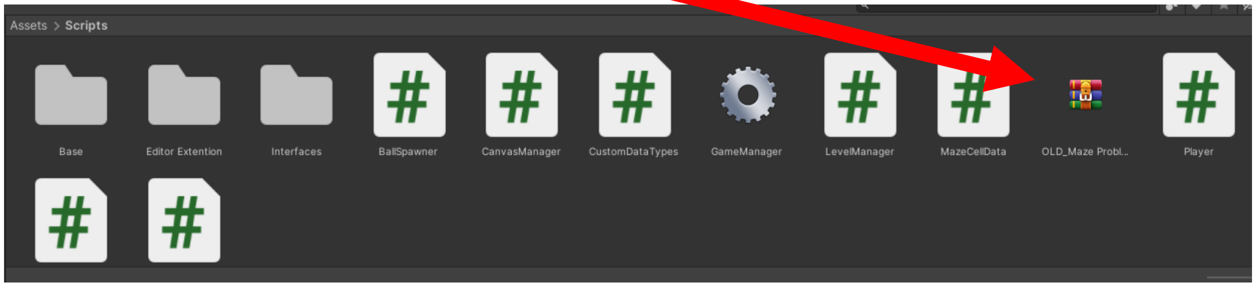
3- Fitness function. The shorter the distance, the better the chromosome.

4- Run Genetic Algorithm.

N-P Algorithm

Implementation Resources:

These implementations can be found in this zip file. Since I altered CustomDataTypes, they need some modifications to work again.


Assets > Scripts
Base
Editor Extention
Interfaces
BallSpawner
CanvasManager
CustomDataTypes
GameManager
LevelManager
MazeCellData
OLD_Maze Probl...
Player