**G3S**

**Version 1.0**

2014

**Rohin Gosling**

**Intellect + NAG**

100k AI Challenge

2014



**Contents**

Game Grid Layout 1

100K AI Challenge - 2014 - State File 2

State File - Character Symbols 2

State Management 3

States 3

Transitions 3

Sensors 3

Neural Network 4

Dynamic Graph 5

Graphics 6

Object Model 8

World 8

Unit 9

Group 10

Unit Command 11

Group Command 12

Sensors 13

Unit State Observation 14

Definitions 15

Eulerian Path 15

Hamiltonian Path 15

100k Challenge 16

Overview 16

Tournament Information 16

Technical specifications and rules 18



# Game Grid Layout



|  |  |
| --- | --- |
| **Figure 1.1**  Entelect 100k Challenge  Custom Bot-vs-Bot Pac Man Layout. | **Figure 1.1**  Neural Network Architecture.  (Autonomously trained, via self-modifying Particle Swarm Optimizer.)  *See Section 7 for Particle Swarm Optimizer architecture.* |
| **Figure 1.3**  Semi-Symmetric Graph Projection of Game Grid. | **Figure 1.2**  Weighted Graph. |

## 100K AI Challenge - 2014 - State File

|  |  |  |
| --- | --- | --- |
| **State File:** initial.state | | |
|  | 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18  0 █████████████████████████████████████████████████████████  1 ███ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ███ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ███  2 ███ ◌ ██████ ▪ █████████ ▪ ███ ▪ █████████ ▪ ██████ ◌ ███  3 ███ ▪ ██████ ▪ █████████ ▪ ███ ▪ █████████ ▪ ██████ ▪ ███  4 ███ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ███  5 ███ ▪ ██████ ▪ ███ ▪ ███████████████ ▪ ███ ▪ ██████ ▪ ███  6 ███ ▪ ▪ ▪ ▪ ███ ▪ ▪ ▪ ███ ▪ ▪ ▪ ███ ▪ ▪ ▪ ▪ ███  7 ████████████ ▪ █████████ ▪ ███ ▪ █████████ ▪ ████████████  8 ████████████ ▪ ███ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ███ ▪ ████████████  9 ████████████ ▪ ███ ▪ ██████ ██████ ▪ ███ ▪ ████████████  10 A ► ▪ ▪ ▪ ███ ███ ▪ ▪ ▪ ◄ B  11 ████████████ ▪ ███ ▪ ██████ ██████ ▪ ███ ▪ ████████████  12 ████████████ ▪ ███ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ███ ▪ ████████████  13 ████████████ ▪ ███ ▪ ███████████████ ▪ ███ ▪ ████████████  14 ███ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ███ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ███  15 ███ ▪ ██████ ▪ █████████ ▪ ███ ▪ █████████ ▪ ██████ ▪ ███  16 ███ ◌ ▪ ███ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ███ ▪ ◌ ███  17 ██████ ▪ ███ ▪ ███ ▪ ███████████████ ▪ ███ ▪ ███ ▪ ██████  18 ███ ▪ ▪ ▪ ▪ ███ ▪ ▪ ▪ ███ ▪ ▪ ▪ ███ ▪ ▪ ▪ ▪ ███  19 ███ ▪ ██████████████████ ▪ ███ ▪ ██████████████████ ▪ ███  20 ███ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ▪ ███  21 █████████████████████████████████████████████████████████  A B C D E F G H I J K L M N O P Q R S |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | ###################  #........#........#  #\*##.###.#.###.##\*#  #.##.###.#.###.##.#  #.................#  #.##.#.#####.#.##.#  #....#...#...#....#  ####.###.#.###.####  ####.#.......#.####  ####.#.## ##.#.####  A...# #...B  ####.#.## ##.#.####  ####.#.......#.####  ####.#.#####.#.####  #........#........#  #.##.###.#.###.##.#  #\*.#...........#.\*#  ##.#.#.#####.#.#.##  #....#...#...#....#  #.######.#.######.#  #.................#  ################### |

## State File - Character Symbols

|  |  |  |
| --- | --- | --- |
| **Character** | **Description** | **Consumption Effect** |
| ‘ ’ | White space. | - No effect. |
| ‘#’ | Wall. | - Collision detection required.  - Penetrating walls results in disqualification. |
| ‘.’ | Pill. | - Earn 1 point. |
| ‘\*’ | Bonus pill. | - Earn 10 points. |
| ‘!’ | Poison pill. ( Inventory: x2 ) | - Re-spawn to centre of maze. (9,10).  - Dropping with empty inventory results in disqualification. |
| ‘A’ or ‘B’ | Players A and B. | - The consumed opponent is re-spawned to the centre of the board.  - Consuming players from the re-spawn point results in disqualification. |

# State Management

## States

* Start
  + Xyz.
* Euler Cycle - Aggressive.
  + Xyz.
* Euler Cycle - Passive.
  + Xyz.
* At Spawn Point
  + Xyz.

## Events

* Opponent poses threat.
  + Xyz.
* Opponent does not pose a threat.
  + Xyz.
* Consumed poison pill.
  + Xyz.
* Consumed by opponent.
  + Xyz.

## Transitions

## Sensors

* Radar
  + Terrain
    - Obstacles (Walls). ( Set of coordinates ).
    - Legal movement space. (White space). ( Set of coordinates ).
  + Pills. ( Set of coordinates ).
  + Bonus pills. ( Set of coordinates ).
  + Poison pills. ( Set of coordinates ).
  + Opponent/s. ( Set of coordinates ).
* Threat Sensor
  + Opponent poses a threat.
* On players move, the Manhattan distance between the player and the opponent is an even number.

* + Opponent is unable to pose a threat.
    - On players move, the Manhattan distance between the player and the opponent is an odd number.
* Inventory
  + Poison pills, x 2.

# Neural Network

using PerceptronValue = double;

enum PerceptronTransferFunction

{

SIGN,

SIGMOID,

GOUSIAN

};

class NeuralNetwork

{

Vector <PerceptronValue> inputLayer; // ( x0 x1 x2 ... xn )

vector <PerceptronLayer> layers; // ( L0 L1 L2 ... Ln )

void Clear ();

PerceptronLayer\* AddLayer ( long perceptronCount );

};

class PerceptronLayer

{

vector <PerceptronUnit> perceptrons; // ( p0 p1 p2 ... pn )

PerceptronValue bias; // b = 1, by default.

double connectivityRatio; // 1.0 = Fully connected.

void Clear ();

void AddPerceptrons ( long perceptronCount );

void SetWeights ( PerceptronValue weightValue );

void RandomizeWeights ( PerceptronValue min, PerceptronValue max );

void DistributeWeights ( double mean, double deviation );

};

class PerceptronUnit

{

PerceptronValue output; // y

vector <PerceptronValue> weightVector; // ( w0 w1 w2 ... wn )

vector <PerceptronValue> inputVector; // ( x0 x1 x2 ... xn )

PerceptronTransferFunction transferFunction; // ʄ ϵ [Sign(xi), Sigmoid(xi), Gousian(xi)]

PerceptronValue Evaluate ();

PerceptronValue Evaluate ( vector <PerceptronValue> inputVector );

};

# Dynamic Graph

using VertexSet = map < string, Vertex >;

using EdgeSet = map < string, Edge >;

using EdgeLength = int;

using EdgeWeight = double;

using VertexValue = double;

using VertexWeight = double;

using ElementColor = vector <double>;

class GraphBase

{

long id; // Unitque element ID.

string name; // Optional element name. Default = “Element” + id.

bool enabled; // true = enabled, false = disabled.

auto tag; // General purpose auxiliry value.

ElementColor enabledColor; // Normal color.

ElementColor disabledColor; // Color used when the element is disabled.

ElementColor selectedEnabledColor; // Color used then an enabled element is selected.

ElementColor selectedDisabledColor; // Color used then a disabeled element Is selected.

}

class Graph : GraphBase

{

VertexSet vertices; // Vertex set: V = {v0,v1,v2,...,vn}

EdgeSet edges; // Edge set: E = {e0(vt,vh),e1(vt,vh),e2(vt,vh),...,en(vt,vh)}

// Vertex functions.

long AddVertex ();

long AddVertex ( string name );

Vertex\* GetVertex ( long id );

Vertex\* GetVertex ( string name );

long SetVertex ( string name, VertexValue value, VertexWeight weight );

string SetVertex ( long id, VertexValue value, VertexWeight weight );

long SetVertexValue ( string name, VertexValue value );

string SetVertexValue ( long id, VertexValue value );

long SetVertexWeight ( string name, VertexWeight weight );

string SetVertexWeight ( long id, VertexWeight weight );

string DeleteVertex ( long id );

Long DeleteVertex ( string name );

// Edge functions.

long AddEdge ();

long AddEdge ( string name );

long AddEdge ( Vertex\* tail, Vertex\* head );

long AddEdge ( string name, Vertex\* tail, Vertex\* head );

Edge GetEdge ( long id );

Edge GetEdge ( string name );

long SetEdge ( string name, Vertex\* tail, Vertex\* head );

long SetEdgeTail ( string name, Vertex\* tail );

long SetEdgeHead ( string name, Vertex\* head );

long SetEdgeLength ( string name, EdgeLength length );

long SetEdgeWeight ( string name, EdgeWeight weight );

long SetEdgeDirection ( string name, bool directed );

String DeleteEdge ( long name );

Long DeleteEdge ( string name );

}

class Edge : GraphBase

{

Vertex\* tail; // Edge = (tail,head).

Vertex\* head; // Edge = (tail,head).

EdgeLength length; // Length of the edge.

EdgeWeight weight; // Weight of the edge.

bool directed; // true = directed, false = undirected.

}

struct Vertex : GraphBase

{

VertexValue value; // Vertex value.

VertexWeight weight; // Vertex weight.

}

# Graphics





# Object Model

## World

**World**

* Name
* Description
* Array of **GameGrid**
* Array of Unit
* Array of Structure

**GameGrid**

* Name
* Description
* Width
* Height
* Array of **Unit**

## Unit

**Unit**

* Name
* Description
* HitPoints
* Armour
* Shield
* Energy
* Kills
* UnitType (Stationary, Mobile)
* CollisionMapShape (Point, Circle, Rectangle)
* Width
* Height
* OriginOffsetX
* OriginOffsetY
* **CollisionMap**
* **TransparencyMap**
* **ColorMap**
* **Array of Weapon**

**Weapon**

* Name
* Description
* WeaponType
* Range
* HitPointDamage
* ArmourDamage
* ShieldDamage
* SplashDamageRadiusX
* SplashDamageRadiusY
* AmmunitionType
* **Array of Ammunition**

**Ammunition**

* Name
* Description
* AmmunitionType
* RangeMultiplier
* HitPointDamageMultiplier
* ArmourDamageMultiplier
* ShieldDamageMultiplier
* SplashDamageRadiusXMultiplier
* SplashDamageRadiusYMultiplier

## Group

**Group**

* Name
* Description
* **Array of Group**
* **Array of Uni**

## Unit Command

**UnitCommand**

* **General**
  + NoOperation
* **Movement**
  + Stop
  + MoveTo ( x, y )
  + MoveToRadius ( x, y, Radius )
  + FollowUnit ( UnitID, Radius )
  + AlignWithCoordinate ( x, y, Angle )
  + AlignWithCoordinateAtRadius ( x, y, Angle, Radius )
  + AlignWithUnit ( UnitID, Angle )
  + AlignWithUnitAtRadius ( UnitID, Angle, Radius )
  + SetHeading (Angle)
  + SetRadialHeadingIn ( x, y )
  + SetRadialHeadingOut ( x, y )
  + HeadingTrackUnitIn ( UnitID )
  + HeadingTrackUnitOut ( UnitID )
* **FireControl**
  + Ceasefire
  + FireDownRange
  + ContinuousFireDownRange
  + AttackCoordinate ( x, y )
  + ContinousAtackCoordinate ( x, y )
  + AttackUnit ( UnitID )
  + ContinourAttackUnit ( UnitID )
  + AttackHeading ( Angle )
  + ContinousAttackHEading ( Angle )
* **Group**
  + LeaveGroup
  + JoinGroup ( GroupID )
  + SetPosition ( PositionID )
  + PromotePosition
  + DemotePosition

## Group Command

**Group Command**

* **General**
  + NoOperation
* **Movement**
  + Stop
  + MoveTo ( x, y )
  + MoveToRadius ( x, y, Radius )
  + FollowUnit ( UnitID, Radius )
  + AlignWithCoordinate ( x, y, Angle )
  + AlignWithCoordinateAtRadius ( x, y, Angle, Radius )
  + AlignWithUnit ( UnitID, Angle )
  + AlignWithUnitAtRadius ( UnitID, Angle, Radius )
  + SetHeading (Angle)
  + SetRadialHeadingIn ( x, y )
  + SetRadialHeadingOut ( x, y )
  + HeadingTrackUnitIn ( UnitID )
  + HeadingTrackUnitOut ( UnitID )
* **FireControl**
  + Ceasefire
  + FireDownRange
  + ContinuousFireDownRange
  + AttackCoordinate ( x, y )
  + ContinousAtackCoordinate ( x, y )
  + AttackUnit ( UnitID )
  + ContinourAttackUnit ( UnitID )
  + AttackHeading ( Angle )
  + ContinousAttackHEading ( Angle )
* **Group**
  + LeaveGroup
  + JoinGroup ( GroupID )
  + SetPosition ( PositionID )
  + PromotePosition
  + DemotePosition
* **Formation**
  + FreeMovement
  + TightLine
  + LooseLine
  + TightBox ( Rows, Columns )
  + LooseBox ( Rows, Columns )

## Sensors

**Sensors**

* Radar
  + Array of Unit
  + Array of Unit. Distance
  + Array of Unit. Radial
  + Array of Unit. Heading
  + Array of Unit. Velocity
  + Array of Unit. TargetLock ( UnitID )
  + Array of UnitAssetDistance ( UnitID, AssetID )
* TerrainScanner ( Range, Resolution )
  + Array of Terrain Feature

## Unit State Observation

**UnitStateObservation**

***Note:***

* ***Subject matter to research:***
  + *Military fire control discipline.*
  + *Military radio protocols and terminology.*
* Health
  + HitPoints
  + Armour
  + Shield
* Situation
  + Position
  + Velocity
  + Heading
* Engagement
  + TargetLock ( UnitID )
  + ReceivingFire ( Range )

# Definitions

## Eulerian Path

* In graph theory, a Eulerian trail (or Eulerian path) is a trail in a graph which visits every edge exactly once.
* A Eulerian circuit (or Eulerian cycle) is an Eulerian trail which starts and ends on the same vertex.

## Hamiltonian Path

* In graph theory, a Hamiltonian path (or traceable trail) is a path in an undirected or directed graph that visits each vertex exactly once.
* A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian path that is a cycle.
* Determining whether such paths and cycles exist in graphs is the Hamiltonian path problem, which is NP-complete.

# 100k Challenge

## Overview

This year's competition sees entrants compete in the classic game of Pacman, with a twist. Players will face-off with an opponent in a pacman-style maze with the goal being to get more points (consume more pills) than your opponent.

Entrants will be divided into two separate pools based on their abilities, with each pool yielding grand-prize winners of the 2014 challenge. This effectively allows beginner programmers and AI enthusiasts to compete against each other with a better chance of making it into the final 4 than previous years. Similarly, the more advanced entrants will pair off against each other in the advanced pool for the grand prize of R100 000.

The competition will see entrants play-off at a special event in September in Melrose Arch in front of gaming and technology media where we will announce our final 4 contestants in each pool. The grand finale will take place live on stage at rAge expo on the 4th of October, where we will be awarding the grand prizes to the winners of both pools. This year, Entelect will be opening up the opportunity for programmers and designers to build their own Graphical User Interfaces (GUI) for the game and will be presenting a special Gooey Award at the rAge expo in October.

## Tournament Information

**Pacman Duel**

This year's competition sees entrants compete in the classic game of Pacman, with a twist. Players will face-off with an opponent in a pacman-style maze with the goal being to get more points (consume more pills) than your opponent.

* Each player will start on opposing sides of the maze (the same maze is used for every game and is a slight modification of the classic pacman maze) and will take turns to move.
* On the board, players will find standard pills and bonus pills, each weighted differently in terms of points.
* To make things exciting, players can send their opponents back to the centre of the maze (respawn) by consuming them.

The objective of the game is to collect more points than your opponent before there no longer any pills left on the maze. It's a race to the finish with only one victor remaining.

**Tournament Pools**

This year's challenge will see contestants split into two pools. Pool A will contain the more advanced entries in the challenge and Pool B will feature the remainder. Each pool will be run as a separate double-elimination tournament, with separate prizes. Entelect will determine the appropriate pool for each entrant based on the ability of the entry. The pool for an entrant will be determined by playing each entry against a set of reference players to determine their rank.

**Prizes**

This year there will be two tournaments, Pool A (advanced) and Pool B (beginner) which will give entrants in the beginner pool a stronger chance at winning the grand prize.

* Pool A (advanced), with a grand prize of R100 000 in cash
* Runner up prizes for final 4 contestants to be announced
* Pool B (beginner) with a grand prize of R50 000 in cash
* Runner up prizes for final 4 contestants to be announced
* Gooey award! The public are encouraged to build their own GUI (Graphical User interface) for the pacman game. The most innovative and impressive GUI will win this award which comes with a cash prize of R15 000.

**Events & Timelines**

* Competition opens on the 14th April 2014
* Competition closes for entries on the 16th of August 2014 and entrants will be notified of their pools by the 1st September
* The play-offs event will be held on the evening of the 13th September at the Fire & Ice Hotel in Melrose Arch. All entrants will be invited to attend this event.
* The finals will be held live at rAge Expo on the main stage on the 11th October.

**Test Harness**

This year Entelect have made the official test harness code available to the public for review. You can view the code and corresponding documentation at the Entelect Challenge Github repository, here.

Binaries will be available for download on the github repositories and will be updated with each modification.

You are encouraged to review the code and submit bug-fixes or suggest modifications. These modifications will be reviewed by the Entelect technical panel and if applicable will be implemented.

## Technical specifications and rules

To – Do

1. **Maze symbols**

An example of a game state file can be found here. The various symbols in the game state file can be interpreted as follows:

Wall - #

Pill - .

Bonus Pill - \*

Poison Pill - !

Players - A, B

5. Technical

Your program must be able to be compiled in either a Java 7 or a .NET 4 environment. (Java or C# are suggested)

The operating system that will be used for the challenge is Windows Server 2012.

Java programs must be compiled using maven and .NET projects must be compiled using msbuild.

The bot is required to be compiled to an executable artifact that accepts a single string parameter. The parameter is a path to a file which represents the state of the game for the players current turn. For example:

c:\challenge\start.bat

The same file will be used to write out the resulting state of the board after the player’s turn has been completed.

The bot is required to produce an output file as per above in the correct format within 4 seconds.

If the bot takes longer than exactly 4 seconds to write the file, the bot will forfeit the match. Be sure to implement the correct safety mechanisms to ensure this rule is not violated.

The bot's program will be terminated after exactly 4 second. Any persistent state or data that you wish to keep across turns will be allowed to be stored in temporary disk files.

The game state file contains lines specifying the entire state of the board at the time the player’s program is called.

An example game.state file is provided below. Please note that the file should be UTF-8 encoded, with Windows format EOL conversion. Each line should be terminated with a carriage return and line feed character as illustrated below. A good tool for visually inspecting this is Notepad++. Go to View->Show Symbol->Show all characters to view CR and LF chars. You can also convert your file to Windows format EOL conversion using Edit->EOL Conversion-> Windows Format and select Encoding->Encode in UTF-8 to ensure UTF-8 encoding.

Download a full example game.state file here. This file contains a clear board with starting points for both you and your opponent.

The output of your program must be written to the same file (game.state) that was supplied in the input.

In addition to the game board required by the competition, the player’s program may output additional files to aid in storing the historical state of the game. These could be necessary in long-term strategies as the player’s program will be terminated after each turn. These files can only be created in the current directory of the application and not necessarily the location that the executable was compiled to. This is described further in the next section.

Your entry must be composed of the following:

Your entry must contain a source folder as well as a start.bat file that launches the program.

Make sure you include a compile.bat file, we’ll use this to automatically compile your program.

Executable binaries for Windows platform (Windows Server 2012)

Ensure that your submissions are less than 5MB in size

readme.txt file which includes instructions for building your project and a brief description of your project structure and strategy.

Ensure that your project is self-contained, i.e all dependencies are bundled with your project and won't need to be downloaded.

Do not hard-code any paths to files or dependencies

6. Start State

7. Test Cases

The test cases are an addition to the rules, and logic in the test harness available on GitHub. The test cases will be useful in testing your bot in the different scenarios that may occur. You can download them here.