

BIRKBECK COLLEGE

MSC COMPUTER SCIENCE PROJECT REPORT

---

# Learning and Video Games: Implementing an Evolutionary Agent

---

*Author:*  
Monty WEST

*Supervisor:*  
Dr. George MAGOULAS

*MSc Computer Science project report, Department of Computer Science  
and Information Systems, Birkbeck College, University of London 2015*

*This report is substantially the result of my own work, expressed in my own  
words, except where explicitly indicated in the text. I give my permission  
for it to be submitted to the JISC Plagiarism Detection Service.*

*The report may be freely copied and distributed provided the source is  
explicitly acknowledged.*



## **Abstract**

Artificial intelligence in video games has long shunned the use of machine learning in favour of a handcrafted approach. However, the recent rise in the use of video games as a benchmark for academic AI research has demonstrated interesting and successful learning approaches. This project follows this research and explores the viability of a game-playing learning AI. Considering previous approaches, an evolutionary agent was created for a platform game based on Super Mario Bros.

The project builds on top of software developed for the Mario AI Competition, which provides the game-engine and agent interface, as well as several other pertinent features. The basic agent was constructed first and a learning framework was built to improve it, utilising a genetic algorithm. The project followed an agile methodology, revisiting design by analysing learning capability.

The aim was to produce an agent that shows meaningful improvement during learning and demonstrated unforeseen behaviours. Ultimately this was achieved. The final learnt agent is able to complete most medium and some high difficulty levels and demonstrated several advanced strategies.

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Concept Definitions . . . . .	7
1.1.1	Intelligent Agents (IAs) . . . . .	7
1.1.2	Rule-based systems . . . . .	7
1.1.3	Biologically Inspired Learning . . . . .	7
1.1.4	Online/Offline Learning . . . . .	9
1.2	Motivation . . . . .	9
1.3	Aim and Objectives . . . . .	10
1.4	Methodology . . . . .	10
1.5	Report Structure . . . . .	13
<b>2</b>	<b>Existing Work</b>	<b>15</b>
2.1	Learning Agents and Game AI Competitions . . . . .	15
2.1.1	The Mario AI Competition . . . . .	16
2.2	Previous Learning Agent Approaches . . . . .	17
2.2.1	Evolutionary Algorithms . . . . .	17
2.2.2	Multi-tiered Approaches . . . . .	18
2.2.3	REALM . . . . .	18
<b>3</b>	<b>Project Specification</b>	<b>21</b>
3.1	Functional Requirements . . . . .	21
3.1.1	Agent . . . . .	21
3.1.2	Level Playing . . . . .	21
3.1.3	Learning . . . . .	21
3.2	Non-functional requirements . . . . .	22
3.3	Major Dependencies . . . . .	22
<b>4</b>	<b>Agent Framework</b>	<b>23</b>
4.1	Design . . . . .	23
4.2	Language and Tools . . . . .	25
4.2.1	Scala . . . . .	25
4.2.2	Maven . . . . .	25
4.3	Implementation . . . . .	26
4.3.1	Perceptions . . . . .	26
4.3.2	Perceiving the Environment . . . . .	28
4.3.3	Observation, Conditions and Actions . . . . .	29
4.3.4	Rules and Rulesets . . . . .	30
4.3.5	Persistence . . . . .	31
4.4	Testing . . . . .	31
4.5	Handcrafted Agents . . . . .	32

<b>5</b>	<b>Level Playing Module</b>	<b>33</b>
5.1	Design . . . . .	33
5.2	Implementation . . . . .	35
5.2.1	Parameter Classes . . . . .	35
5.2.2	Persistence . . . . .	35
5.2.3	The Episode Loop . . . . .	36
5.3	Modifying the Benchmark . . . . .	38
5.3.1	Enemy Generation . . . . .	39
5.3.2	Pit Generation . . . . .	39
5.4	Testing . . . . .	39
5.5	Comparator Task . . . . .	40
<b>6</b>	<b>Learning</b>	<b>41</b>
6.1	Evolutionary Approach . . . . .	41
6.2	The ECJ Library . . . . .	42
6.3	Implementation . . . . .	45
6.3.1	Modifying the ECJ Library . . . . .	45
6.3.2	Species . . . . .	46
6.3.3	Mutation . . . . .	48
6.3.4	Evaluation . . . . .	50
6.3.5	Statistics . . . . .	51
6.4	Testing . . . . .	52
6.5	Running . . . . .	52
6.6	Parameters and Revisions . . . . .	53
<b>7</b>	<b>Results</b>	<b>55</b>
7.1	Learning Data . . . . .	55
7.1.1	Improvement over Time . . . . .	55
7.1.2	Generation Variance . . . . .	55
7.1.3	Population Variance . . . . .	57
7.2	Handcrafted vs. Learnt Agents . . . . .	59
7.2.1	Learning Evaluation Task . . . . .	59
7.2.2	Comparator Task . . . . .	60
7.3	Learnt Agent Analysis . . . . .	61
7.3.1	Rule usage . . . . .	61
7.3.2	Behaviours . . . . .	63
<b>8</b>	<b>Project Evaluation</b>	<b>71</b>
8.1	Agent Framework . . . . .	71
8.2	Level Playing . . . . .	72
8.3	Learning Process . . . . .	74
8.4	Methodology . . . . .	75

<b>Appendices</b>	<b>76</b>
A Perceptions . . . . .	76
B Handcrafted Agent Rulesets . . . . .	77
C Full Level and Evaluation Options . . . . .	78
D Comparator Task Options . . . . .	79
E LEMMEL Learning Parameter File . . . . .	80
F Full Source Code . . . . .	83

# 1 Introduction

Artificial intelligence (AI) is a core tenant of video games, traditionally utilised as adversaries or opponents to human players. Likewise, game playing has long been a staple of AI research. However, academic research has traditionally focused mostly on board and card games and advances in game AI and academic AI have largely remained distinct.

The first video game opponents were simple discrete algorithms, such as the computer paddle in *Pong*. In the late 1970s, video game AIs became more advanced, utilising search algorithms and reacting to user input. In *Pacman*, the ghost displayed distinct personalities and worked together against the human player [1]. In the mid 1990s, approaches became more ‘agent’ based. Finite State Machines (FSMs) emerged as a dominant game AI technique, as seen in games like *Half-Life* [2]. Later, in the 2000s, Behaviour Trees gained pre-eminence, as seen in games such as *F.E.A.R.* [3] and *Halo 2* [4]. These later advances borrowed little from contemporary development in academic AI and remained localised to the gaming industry.

In the last ten years, with an increase in processing power and developments in the complexity of games, many academic techniques have been harnessed by developers [5]. For example, Monte Carlo Tree Search techniques developed in Go AI research have been used in *Total War: Rome II* [6]. In 2008’s *Left 4 Dead*, Player Modelling was used to alter play experience for different users [7, p. 10]. Furthermore, AI and related techniques are no longer only being used as adversaries. There has been a rise in intelligent Procedural Content Generation in games in recent years, in both a game-world sense (for example *MineCraft* and *Terraria*) and also a story sense (for example *Skyrim*’s Radiant Quest System) [8].

However, machine learning has yet to find a meaningful contribution to the world of video gaming, despite being a staple of academic research into AI, especially in robotics and board games. High complexity, small datasets and time constraints greatly hinder the effective implementation of learning techniques in the industry.

Conversely, commercial games have recently enjoyed more consideration in academic research. Games such as *Ms. Pac Man*, *Starcraft*, *Unreal Tournament*, *Super Mario Bros.* and open-source counterparts *TORCS* [28] and *Cellz* [10] have been at the centre of recent competitions and papers [11] [12]. These competitions tend to focus on agent-based game-playing AI, with many entrants adopting an evolutionary learning approach. This research could have applications as AI ‘competitors’ to human players, which is especially relevant to the racing, FPS and RTS genres.

This project will combine the notions of game-playing agents, evolutionary learning and traditional video game AI. By considering similar existing work, an evolutionary agent will be produced that learns to effectively play a 2D platforming game.

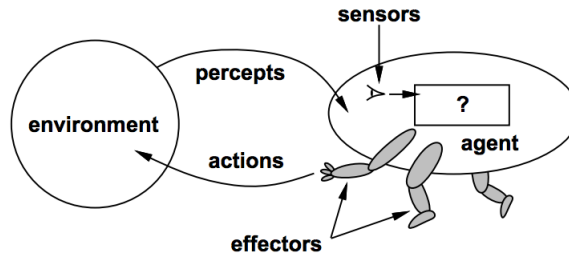


Figure 1: Illustration of an intelligent agent, taking from [16, p. 32]

## 1.1 Concept Definitions

At this point it is useful to introduce some high level descriptions/definitions of some key concepts that will be used in this report.

### 1.1.1 Intelligent Agents (IAs)

An intelligent agent is an entity that uses **sensors** to perceive its **environment** and acts based on that perception through **actuators** or **effectors**. In software, this is often realised as an autonomous program or module that takes its perception of the **environment** as input and returns **actions** as output. Figure 1 shows the basic structure of an intelligent agent. [15, p. 34]

### 1.1.2 Rule-based systems

A rule-based system decides **actions** from **inputs** as prescribed by a **ruleset** or **rule base**. A **semantic reasoner** is used to manage the relationship between input and the ruleset. This follows a **match-resolve-act** cycle, which first finds all rules matching an input, chooses one based on a conflict strategy and then uses the rule to act on the input, usually in the form of an output. [17, pp. 28-29]

### 1.1.3 Biologically Inspired Learning

Several computational learning approaches have derived their inspiration from learning in animals and humans. Two such approaches are relevant to this project: Reinforcement learning, strategy modelled on simplistic interpretation of how animals learn behaviour from their environment [13, s. 1.2]; and evolutionary computation, algorithms that apply Darwinian principles of evolution [14].

## Reinforcement Learning

A reinforcement learning agent focuses on a learning problem, with its goal to maximise **reward**. Given a current **state**, the agent chooses an **action**



available to it, which is determined by a **policy**. This action maps the current **state** to a new **state**. The **transition** is then evaluated for its **reward**. This **reward** often affects the **policy** of future iterations, but **policies** may be stochastic to some level. [13, s. 1.3]

## Evolutionary Computation

Evolutionary computation encompasses many learning algorithms, two of which are described in detail below. The process looks to optimise a data representation of the problem through random variation and selection in a **population**. It commonly employs techniques similar to survival, breeding and mutation found in biological evolutionary theory. [14]

### Genetic Algorithms (GAs)

Genetic Algorithms are a subset of evolutionary computation. They model the solution as a **population** of **individuals**. Each **individual** has a set of **genes** (a **genome**), which can be thought of as simple pieces of analogous information (most often in the form of bit strings). Each **individual** is assessed by some **fitness function**. This assessment can be used to cull the **population**, akin to survival of the fittest, or to increase the individual's chance of influencing the next **population**. The new **population** is created by **breeding**, using a combination of the following: **crossover** of the **genome** from two (or more) **individuals** (akin to sexual reproduction), **mutation** of the **genes** of one **individual** (akin to asexual reproduction) and **re-ordering** of the **genes** of one **individual**. Each new **population** is called a **generation**. [18, p. 7]

### Evolution Strategies (ESes)

Evolution Strategies are another example of evolutionary computation. They differ from standard Genetic Algorithms by using **truncation selection** before breeding. The top  $\mu$  individuals of the population are chosen (usually by fitness) and bred to create  $\lambda$  children. ES notation has the following form:  $(\mu/\rho \nmid \lambda)$ .  $\rho$  denotes the number of individuals from  $\mu$  used in the creation of a single  $\lambda$ , (i.e. number of parents of each child) this report will only consider the case  $\rho = 1$ . The  $+$  and  $,$  are explained below: [19, p. 6-10] [40, s. 4.1.2]

$(\mu , \lambda)$  Denotes an ES that has a population size of  $\lambda$ . The top  $\mu$  individuals are taken from the  $\lambda$  in generation  $g - 1$ , which then produce  $\lambda$  children for generation  $g$ . This is done by creating  $\lambda/\mu$  clones of each  $\mu$  and then mutating them individually.

**$(\mu + \lambda)$**  Differs from the  **$(\mu, \lambda)$**  variant by adding the  $\mu$  individuals chosen from generation  $g-1$  to the new generation  $g$  after the mutation phase. Hence the population size is  $\lambda + \mu$ .

#### 1.1.4 Online/Offline Learning

**Offline** An offline (or batch) learner trains on an entire dataset before applying changes.

**Online** A online learner reacts/learns from data immediately after each datapoint.

## 1.2 Motivation

Ventures in utilising learning in commercial video games have been limited and largely ineffectual. Game development works on strict cycles and there are limited resources to invest into AI research, especially if the outcome is uncertain. Furthermore, one player playing one game produces a very small data set, making learning from the player challenging. [21]

However, there are many reasons why good execution of these techniques is desirable, especially biologically inspired learning. Humans must learn and react to environments and scenarios during games, based purely on their perception of the game (and not its inner working). Having non-playable characters do the same may produce a more believable, immersive and relatable AI. Secondly, such learning algorithms produce agents that can respond well in new situations (over say FSMs or discrete logic), making new content easy to produce or generate. Lastly, modern games have large and diverse player bases, having a game that can respond and personalise to a specific player can help cater to all. [11, p. 7, p. 13]

Despite the lack of commercial success, video games can act as great benchmark for learning agents. Playing games (as a human) naturally has some degree a learning curve. They are designed to challenge humans, and will therefore challenge learning methods, particularly those inspired by biological processes. Also, games require quick reactions to stimulus, something not true of traditional AI challenges such as board games. In addition, most games have some notion of scoring, suitable for a fitness function. Lastly, they are generally accessible to students, academics and the general public alike. [11, p. 9] [12, p. 1] [25, p. 2]

As such, games have now been utilised in several academic AI competitions. These competitions are at the forefront of research and development into learning techniques in video games, and will be explored in more detail in Section 2.1.

Exploring the use of learning techniques for use in video games is a challenging and eminent area of research, with interest from both the video

game and computation intelligence communities. This project is motivated by this fact and influenced by the variety of previous approaches taken in these competitions and the unexpected results they produced.

### 1.3 Aim and Objectives

The aim of the project is to explore the use of behavioural learning techniques in creating a game-playing agent-based AI. This will be achieved by producing an intelligent agent, developed by an evolutionary algorithm, that plays a 2D side-scrolling platform game.

#### Objectives

1. **Design**

Influenced by previous approaches, design an agent and learning procedure that can demonstrate meaningful improvement during learning.

2. **Implementation**

Implement agent and learning designs in a modular, customisable and test-driven manner, utilising external libraries and features of the game software.

3. **Testing**

Provide significant test coverage to the functionality of the implementation.

4. **Learning**

Grant the agent a large and diverse testbed from which to learn as well as ample time and resources to do so.

5. **Evaluation**

Evaluate both the competence and learning capability of the agent and compare to alternative approaches.

### 1.4 Methodology

The two central components to this project, the agent and the learning process, will be designed, implemented and tested as two separate modules. This presents a clean separation of concerns. The agent module will be completed first, followed by the learning module. A third module will also be created, which will allow the agent to play generated levels and receive a score.

It is foreseeable that certain parts of the project will be exploratory and as such a redesign of either the agent or learning module may be required. Therefore, it will be important to take an agile and iterative approach to production, revisiting past objectives if need be.

The open-source Mario benchmark software<sup>1</sup> (used in the Mario AI Competition) will form the initial codebase for this project. It is written in Java and contains the game-engine and useful features such as level generation and level playing classes.

I intend to use Scala as the main language of this project as I believe that its functional and concurrent features will suit the problem. Scala and Java are widely compatible (as they both compile JVM bytecode) so integration should not have a significant impact on project time. However, the codebase will be assessed for the suitability of this approach, and Java 8 will be used if Scala proves to be counterproductive.

Furthermore, the software will be assessed for the inclusion of build tools, such as *sbt* and *Maven*; testing frameworks, such as *ScalaMock* and *Mockito* and logging, such as *log4j* and Scala’s inbuilt logging package.

### Objective 1: Design

The design of both the agent and learning module will follow the approach adopted in previous work regarding the REALM agent, which is discussed in detail in Section 2.2.3.

As in previous work, the agent module developed for the project will be a rule-based system that maps environment based conditions to explicit key presses. However, the proposed approach will differ from the REALM agent by exploring several additional perceptions of the available sensory information. These perceptions will be limited to ‘visible’ environment (e.g. the direction Mario is moving, presence of enemies etc.). They will be measured and distilled into a collection of simple observations.

The learning module will utilise an offline genetic algorithm as seen in previous work, which will be described later in Section 2.2.3. Agents will be individuals, with rulesets as genomes and fitness will be determined by playing a series of generated levels. However, different approaches to mutation, crossover and reordering will be explored, as well as careful consideration of calculating fitness from level playing statistics.

### Objective 2: Implementation

The agent module will conform to the *Agent* interface included in the benchmark. This will allow the agent to be used in other areas of the benchmark software, such as the game-playing and evaluation classes. Each agent will be defined by its ruleset, the contents of which will determine its behaviour. The agent will implement a semantic reasoner for the ruleset, returning an action as prescribed by the chosen rule.

The agent will gather its sensory information from the game’s *Environment* class, which reports visible properties of the level scene. If necessary

---

<sup>1</sup>Available at <http://www.marioai.org/gameplay-track/getting-started>

this class will be extended to include any missing ‘visible’ data. As discussed in 1.4, this will be simplified to a list of observations and compared against the agent’s ruleset.

The learning module will utilise an external library alongside the level playing module. Many evolutionary computation libraries exist for Java (and therefore Scala), *ECJ* [39] and *JGAP* [42] will both be considered for use in this project.

The implementation will manage the integration of this library with both the agent and level playing modules. The algorithm will evolve a simple data structure representation of rulesets, inject them into agents and assess those agents by having them play several carefully parametrised levels. Statistics returned from these levels will form the base variables of the fitness function, with multipliers being configurable. To aid improvement of the learning process, these parameters will be held externally and fitness values will be logged.

### **Objective 3: Testing**

Test coverage of the agent module will be handled by black-box testing. Unit tests will be written to test the semantic reasoner and the environment observation methods.

Due to the stochastic nature of genetic algorithms, testing of the learning module will be limited. However, the breeding can be tested by verifying that children stay within the boundaries. Some evolutionary libraries, such as JGAP, provide several unit tests. If available, these will become the primary source of test coverage for the module.

### **Objective 4: Learning**

Optimising the learning parameters (including the parameters of the breeding phase, fitness function and level playing) will be an important stage of the project. Assessment of the parameters used in previous agents such as REALM (discussed in 2.2.3) D. Perez et al. [36] and Agent Smith [44] will inform the initial values. Learning module logs will subsequently be analysed for improvements. For example, if there is a lot of variation in fitness then perhaps mutation should be restricted, or if the average fitness does not eventually level out then the further generations should be created.

The design of the agent can also influence the effectiveness of the learning algorithm. The size of the search space is determined by the conditions and actions of the rulesets, the reduction of which could improve evolutionary capability. Hence, the learning phase of the project may inform a redesign of the agent, which is one of the main reasons why this project will take an agile approach.

The learning itself is likely to be a time consuming, computationally heavy procedure. To assist in providing ample resources to this process the project will have access to two 8 core servers, as well as a laptop with an Intel i7 processor.

## **Objective 5: Evaluation**

On the conclusion of the learning process, the best/final agent will be extracted and evaluated. This will be done by using the level playing module. The agent will go through an extensive set of levels, based on the approach taken by the 2010 Mario AI Competition.

The primary comparison will be with a handcrafted ruleset, which will assess the significance of the agent evolution. Other comparisons can be made against agents that are included as examples in the benchmark, such as the *ForwardJumpingAgent* that was used for similar comparisons in the 2009 competition, as well as other entrants into the competition [26, p. 7].

The second part of this objective is to evaluate the learning procedure. Figures such as average and maximum generation fitness can provide an insight into the effectiveness of the genetic algorithm. Furthermore, a baseline for these values can be provided by having the handcrafted agent play the levels alongside the evolving agent. The final evaluation report will provide an analysis of these figures.

## **1.5 Report Structure**

This report will cover previous approaches to the project’s aim; the design, implementation of the agent and learning process; and evaluate both the results and the project as a whole.

Section 2 details existing work, focusing on relevant entrants in recent Game AI competitions. It will consider their approaches to learning and agent design for both effectiveness and significance to learning in video games. It will look in particular at the Mario AI Competition and the winner entrant in 2010, the REALM agent.

Section 3 will cover the project’s specification, discussing functional and non-functional requirements. It will also include a description of the major dependencies that influenced project design.

Section 4 will explain the design, implementation and testing of the rule-based agent. It will demonstrate how the agent was built to enable evolution by a genetic algorithm, as well as how it perceives its environment and chooses an action. Reasons for the choice of project language and build tools are also included here.

Section 5 will detail the development of the level playing module. This contains an account of the modifications that had to be made to the game

engine software. It covers the extension to the game engine and how it was designed and implemented with a view to parametrisation and level variety.

Section 6 explains the choice of genetic algorithm and the basic parameters used. It also describes extensions made to the learning library to allow it to effectively evolve the agent in an easily customisable and observable fashion. Lastly, it details how specific mutation and fitness parameters were chosen in response to initial learning runs in order to improve the process.

Section 7 presents the data gathered during the learning run(s). Using this data, it studies the effectiveness of the learning algorithm by examining metrics such as fitness increase over generations and variance. It also analyses the learnt agent(s), highlighting interesting behaviour and drawing comparisons with handcrafted agents.

Section 8 evaluates the three major portions of the project. Firstly, it considers the agent framework, evaluating it on complexity, speed and capability, and offering possible improvements. It then discusses the positives and negatives of the level playing module, as well as issues that were left unaddressed. Subsequently, it examines the choice of learning algorithm and parameters, with a view to future revisions and additions. Lastly, it considers the effectiveness of the project methodology as a whole.

Genre	Game	Description
Racing	TORCS (Open-source) [28]	<b>The Simulated Car Racing Competition</b> Competitors enter agent drivers, that undergo races against other entrants which include qualifying and multi-car racing. The competition encourages the use of learning techniques. [29]
First Person Shooter (FPS)	Unreal Tournament 2004	<b>The 2K BotPrize</b> Competitors enter ‘bots’ that play a multi-player game against a mix of other bots and humans. Entrants are judged on Turing test basis, where a panel of judges attempt to identify the human players. [30]
Real Time Strategy (RTS)	Starcraft	<b>The Starcraft AI Competition</b> Agents play against each other in a 1 on 1 knockout style tournament. Implementing an agent involves solving both micro objectives, such as path-planning, and macro objectives, such as base progression. [31]
Platformer	Infinite Mario Bros (Open-source)	<b>The Mario AI Competition</b> Competitors submit agents that attempt to play (as a human would) or create levels. The competition is split into ‘tracks’, including Gameplay, Learning, Turing and Level Generation. In Gameplay, each agent must play unseen levels, earning a score, which is compared to other entrants. [25]

Table 1: This table summarises some recent game AI competitions [27]

## 2 Existing Work

### 2.1 Learning Agents and Game AI Competitions

Over the last few years, several game based AI competitions have been run, over a variety of genres. These competitions challenge entrants to implement an agent that plays a game and is rated according to the competitions specification. They have attracted both academic [25, p. 2] and media interest [12, p. 2]. The competitions tend to encourage the use of learning techniques, hence the recent publication of several interesting papers concerning the application of biologically inspired learning agents in video games. Approaches tend to vary widely, modelling and tackling the problem differently and adapting traditional techniques in previously unseen ways. [25, p. 11]

Some brief details of the competitions which are of relevance to this project are compiled in to Table 1. The Mario AI Competition is also explored in more detail below.



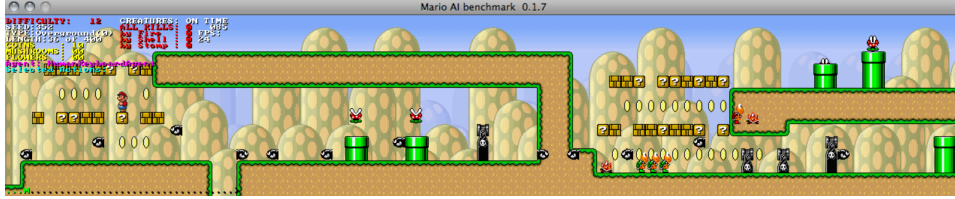


Figure 2: Example of part of a level in the Mario AI Competition’s benchmark software. Taken from [25, p. 6]

### 2.1.1 The Mario AI Competition

The Mario AI Competition, organised by Sergey Karakovskiy and Julian Togelius, ran between 2009-2012 and used an adapted version of the open-source game Infinite Mario Bros. From 2010 onwards the competition was split into four distinct ‘tracks’. We shall focus on the unseen Gameplay track, where agents play several unseen levels as Mario with the aim to finish the level (and score highly). [12] [25]

#### Infinite Mario Bros.

Infinite Mario Bros (IMB) [32] is an open-source clone of Super Mario Bros. 3, created by Markus Persson. The core gameplay is described as a *Platformer*. The game is viewed side-on with a 2D perspective. Players control Mario and travel left to right in an attempt to reach the end of the level (and maximise score). The screen shows a short section of the level, with Mario centred. Mario must navigate terrain and avoid enemies and pits. To do this, Mario can move left and right, jump, duck and speed up. Mario also exists in 3 different states, *small*, *big* and *fire* (the latter of which enables Mario to shoot fireballs), accessed by finding powerups. Touching an enemy (in most cases) reverts Mario to a previous state. Mario dies if he touches an enemy in the *small* state or falls into a pit, at which point the level ends. Score is affected by how many coins Mario has collected, how many enemies he has killed (by jumping on them or by using fireballs or shells) and how quickly he has completed the level. [25, p. 3]

An example of a typical level in IMB can be found in Figure 2.

#### Suitability to Learning

The competitions adaptation of IMB (known henceforth as the ‘benchmark’) incorporates a parameterised level generator and allows for the game to be sped-up by removing the reliance on the GUI and system clock. This makes it a great testbed for reinforcement learning. The ability to learn from large sets of diverse data makes learning a much more effective technique. [25, p. 3]

Besides that, the Mario benchmark presents an interesting challenge for learning algorithms. Despite only a limited view of the “world” at any one time the state and observable space is still of quite high-dimension. Though not to the same extent, so too is the action space. Any combination of five key presses per timestep gives an action space of  $2^5$  [25, p. 3]. Consequently, part of the problem when implementing a learning algorithm for the Mario benchmark is reducing these search spaces. This is the topic of papers by Handa and Ross and Bagnell [34], who separately addressed this issue in their papers [33] and [34] respectively.

Lastly, there is a considerable learning curve associated with Mario. The simplest levels could easily be solved by agents hard coded to jump when they reach an obstruction, whereas difficult levels require complex and varied behaviour. For example, traversing a series of pits may require a well placed series of jumps, or passing a group of enemies may require careful timing. Furthermore, considerations such as score, or the need to backtrack from a dead-end greatly increase the complexity of the problem. [25, p. 3, p. 12]

## 2.2 Previous Learning Agent Approaches

Agent-based AI approaches in commercial games tend to focus on finite state machines, behaviour trees and rulesets, with no learning component. Learning agents are more prevalent in AI competitions and academia, where it is not only encouraged, but viewed as an interesting research topic [12, p. 1]. Examples from both standpoints are compiled in Table 2.

### 2.2.1 Evolutionary Algorithms

In Section 1.1 we presented some basic concepts of evolutionary and genetic computing. This approach is a common choice of learning methods used in game-playing agents. D. Perez et al. note in their paper [36, p. 1] that evolutionary algorithms are particularly suitable for video game environments:

*‘Their stochastic nature, along with tunable high- or low-level representations, contribute to the discovery of non-obvious solutions, while their population-based nature can contribute to adaptability, particularly in dynamic environments.’*

The evolutionary approach has been used across several genres of video games. For example, *neuroevolution*, a technique that evolves neural networks, was used in both a racing game agent (by L. Cardamone [29, p. 137]) and a FPS agent (by the UT $\wedge$ 2 team [30]). Perhaps the most popular approach was to use genetic algorithms (GAs) to evolve a more traditional game AI agent. R. Small used a GA to evolve a ruleset for a FPS agent [44], T. Sandberg evolved parameters of a potential field in his Starcraft agent [38], *City Conquest’s* in-game AI used an agent-based GA-evolved build

plan [21] and D. Perez et al. used a grammatical evolution (a GA variant) to produce behaviour trees for a Mario AI Competition entry [36].

### 2.2.2 Multi-tiered Approaches

Several of the most successful learning agents take a multi-tiered approach. By splitting high-level behaviour from low-level actions agents can demonstrate a more interesting, and even human-like, performance. For example, COBOSTAR, an entrant in the 2009 Simulated Car Racing Competition, used offline learning to determine high-level parameters such as desired speed and angle alongside a low-level crash avoidance module [29, p. 136]. UT<sup>2</sup> used learning to give their FPS bot broad human behaviours and a separate constraint system to limit aiming ability [30]. Overmind, the winner of the 2010 Starcraft Competition, planned resource use and technology progression at a macro level, but used A\* search micro-controllers to coordinate units [9].

One learning agent that successfully utilised both an evolutionary algorithm and a multi-tiered approach is the Mario agent REALM, which is explored in more detail below.

### 2.2.3 REALM

The REALM agent, developed by Slawomir Bojarski and Clare Bates Congdon, was the winner of the 2010 Mario AI competition, in both the unseen and learning Gameplay tracks. REALM stands for **R**ule Based **E**volutionary **C**omputation **A**gent that **L**earns to Play **M**ario. REALM went through two versions (V1 and V2), with the second being the agent submitted to the 2010 competition.

#### Rule-based

Each time step REALM creates a list of binary observations of the current scene, for example IS\_ENEMY\_CLOSE\_LOWER\_RIGHT and IS\_PIT\_AHEAD. Conditions on observations are mapped to actions in a simple ruleset. These conditions are ternary (either TRUE, FALSE or DONT\_CARE) [35, p. 85]. A rule is chosen that best fits the current observations, with ties being settled by rule order, and an action is returned [35, p. 86].

Actions in V1 are explicit key-press combinations, whereas in V2 they are high-level plans. These plans are passed to a simulator, which reassesses the environment and uses A\* to produce the key-press combination. This two-tier approach was designed in part to reduce the search space of the learning algorithm. [35, pp. 85-87]

## Learning

REALM evolved a ruleset using an evolutionary strategy over 1000 generations. The best performing rule set from the final generation was chosen to act as the agent for the competition. Hence, REALM is an agent focused on offline learning. [35, pp. 87-89]

Populations have a fixed size of 50 individuals, with each individual's genome being a ruleset. Each rule represents a gene and each individual has 20. Initially rules are randomised, with each condition having a 30%, 30%, 40% chance to be TRUE, FALSE or DONT\_CARE respectively.

Individuals are evaluated by running through 12 different levels. The fitness of an individual is a modified score, averaged over the levels. Score focuses on distance, completion of level, Mario's state at the end and number of kills. Each level an individual plays increases in difficulty. Levels are predictably generated, with the seed being recalculated at the start of each generation. This is to avoid over-fitting and to encourage more general rules.

REALM used the  $(\mu + \lambda)$  variant ES, with  $\mu = 5$  and  $\lambda = 45$  (i.e. the best 5 individuals are chosen and produce 9 clones each). Offspring are exposed to: **Mutation**, where rule conditions and actions may change value; **Crossover**, where a rule from one child may be swapped with a rule from another child<sup>2</sup> and **Reordering**, where rules are randomly reordered. These occur with probabilities of 10%, 10% and 20% respectively [35, p. 88]. Unfortunately, the method employed to perform these operations is not clearly explained in the REALM paper [35].

## Performance

The REALM V1 agent saw a larger improvement over the evolution, but only achieved 65% of the V2 agent's score on average. It is noted that V1 struggled with high concentrations of enemies and large pits. The creators also assert that the V2 agent was more interesting to watch, exhibiting more advanced and human-like behaviours. [35, pp. 89-90]

The ruleset developed from REALM V2 was entered into the 2010 unseen Gameplay track. It not only scored the highest overall score, but also the highest number of kills and was never disqualified (by taking too long to decide on an action). Competition organisers note that REALM dealt with difficult levels better than other entrants. [25, p. 10]

---

<sup>2</sup>This is similar to a  $(\mu/\rho + \lambda)$  ES approach with  $\rho = 2$ , but crossover occurs in the mutation phase and between all children, rather than specifically with children from another parent.

<b>Name</b>	<b>Game/Competition</b>	<b>Approach</b>
M. Erickson [25]	2009 Mario AI Competition	A crossover heavy GA to evolve an expression tree.
E. Speed [26]	2009 Mario AI Competition	GA to evolve grid-based rulesets. Ran out of memory during the competition.
S. Polikarpov [26, p. 7]	2009-10 The Mario AI Competition	Ontogenetic reinforcement learning to train a neural network with action sequences as neurons.
REALM [35]	2010 Mario AI Competition	GA to evolve rulesets mapping environment to high-level behaviour.
D. Perez et al [36]	2010 Mario AI Competition	Grammatical evolution with a GA to develop behaviour trees.
FEETSIES [37]	2010 Mario AI Competition	“Cuckoo Search via Lévy Flights” to develop a ruleset mapping an observation grid to actions.
COBOSTAR [29, p. 136]	2009 Simulated Car Racing Competition	Covariance matrix adaptation evolution strategy to map sensory information to target angle and speed. Online reinforcement learning to avoid repeating mistakes.
L. Cardamone [29, p. 137]	2009 Simulated Car Racing Competition	Neuroevolution to develop basic driving behaviour.
Agent Smith [44]	Unreal Tournament 3	GAs to evolve very simple rulesets, which determine basic bot behaviour.
UT <sup>2</sup> [30]	2013 2K Botprize	Neuroevolution with a fitness function focused on being ‘human-like’.
T. Sandberg [38]	Starcraft	Evolutionary algorithms to tune potential field parameters.
Berkeley Overmind [9]	The Starcraft AI Competition	Reinforcement learning to tune parameters for potential fields and A* search.
In-game Opponent AI [21]	City Conquest	GAs to evolve build plans with fitness measured in a 1-on-1 AI match.
In-game Creature AI [22]	Black & White	Reinforcement Learning applied to a neural network representing the creatures desires.
In-game Car AI [23]	Project Gotham Racing	Reinforcement learning to optimise racing lines.

*Table 2: Biologically inspired learning, agent-based approaches to game playing AI*

## 3 Project Specification

### 3.1 Functional Requirements

Functionally, the project can be split into three parts: the agent framework, which is responsible for gathering sensory information from the game and producing an action; level generation and playing, which is responsible for having an agent play generated levels with varying parameters; and the learning module, which will apply a genetic algorithm to a representation of an agent, with an aim to improving its level playing ability.

#### 3.1.1 Agent

The agent framework will implement the interface supplied in the Mario AI benchmark; receive the game *Environment* and produce an *Action*. It must be able to encode individual agents into a simple format (e.g. a bit string or collection of numbers). Additionally, it should be able to encode and decode agents to and from external files.

The framework will be assessed in three ways. Firstly on its complexity, keeping the search space of the encoded agent small is important for the learning process. Secondly on its speed, the agent must be able to respond within one game tick. Thirdly on its capability, the framework must facilitate agents that can complete the easiest levels, and attempt the hardest ones. It is also required that human created agent encoding(s) be written (within the framework) to assist this assessment.

#### 3.1.2 Level Playing

The level generation and playing module must be able to generate and play levels using an agent, producing a score on completion. It should extend the existing framework included in the Mario AI Benchmark software. Furthermore, it should be able to read parameters for generation and scoring from an external file.

The module will be evaluated on the level of variety in generated levels and the amount of information it can gather in order to score the agent.

#### 3.1.3 Learning

The learning module should utilise a genetic algorithm to evolve an agent (in encoded form). It should also ensure that as many as possible of the parameters that govern the process can be held in external files, this includes overall strategy as well fine grained detail (e.g. mutation probabilities and evaluation multipliers). Where impossible or inappropriate to hold such parameters externally, it must be able to read them dynamically from their governing packages, for example boundaries on the agent encoding should

be loaded from the agent package, rather than held statically in the learning module. It must have the facility to report statistics from learning runs, as well as write out final evolved agents.

The learning module will also be assessed on three counts. Firstly, learning should not run for too long, granting the freedom to increase generation count or adjust parameters. Secondly, the learning process should demonstrate a meaning improvement to the agent over generations as this demonstrates an effective genetic algorithm. Thirdly, the final evolved agent will be assessed, using the level playing package. It will be tested against the handcrafted agents and analysed for behaviours and strategies not considered during their creation.

### 3.2 Non-functional requirements

Both the level playing module and the agent framework should not prevent or harm thread safety, allowing multi-threading in the learning module. Each part should be deterministic, i.e. if given the same parameter files will always produce the same results. Lastly, the entire project must have the ability to be packaged and run externally.

### 3.3 Major Dependencies

The project has two major dependencies: a game engine and a learning library. Their selection influenced the design of all aspects of the project and hence are included here.

As previously mentioned, the project will extend the Mario AI Benchmark. As previously discussed in Section 2.1.1 the Mario AI Benchmark is an open-source Java code-base, build around a clone of the game Super Mario Bros. 3. It was chosen for its aforementioned suitability to learning and for its other pertinent features, including an agent interface, level playing and generation package and other useful additions (e.g. the ability to turn off the reliance on the system clock).

The use of the Mario AI Benchmark restricts the choice of language (discussed further in Section 4.2.1) to those that can run on the JVM. Hence, ECJ was chosen as the learning library as it is written in Java and is available as both source-code and packaged jar. Furthermore, ECJ fit the project specification very well: it provides support for GAs and ESes, its classes and settings are decided at runtime from external parameter files, it is highly flexible and open to extension, and has facilities for logging run statistics.

Although the intention was to include these dependencies as packaged jars, modification of their source code was necessary (which was available on academic licences). This code was modified in Java, packaged and included as dependencies in the main project. Details of the modifications can found in Sections 5.3 and 6.3.1.

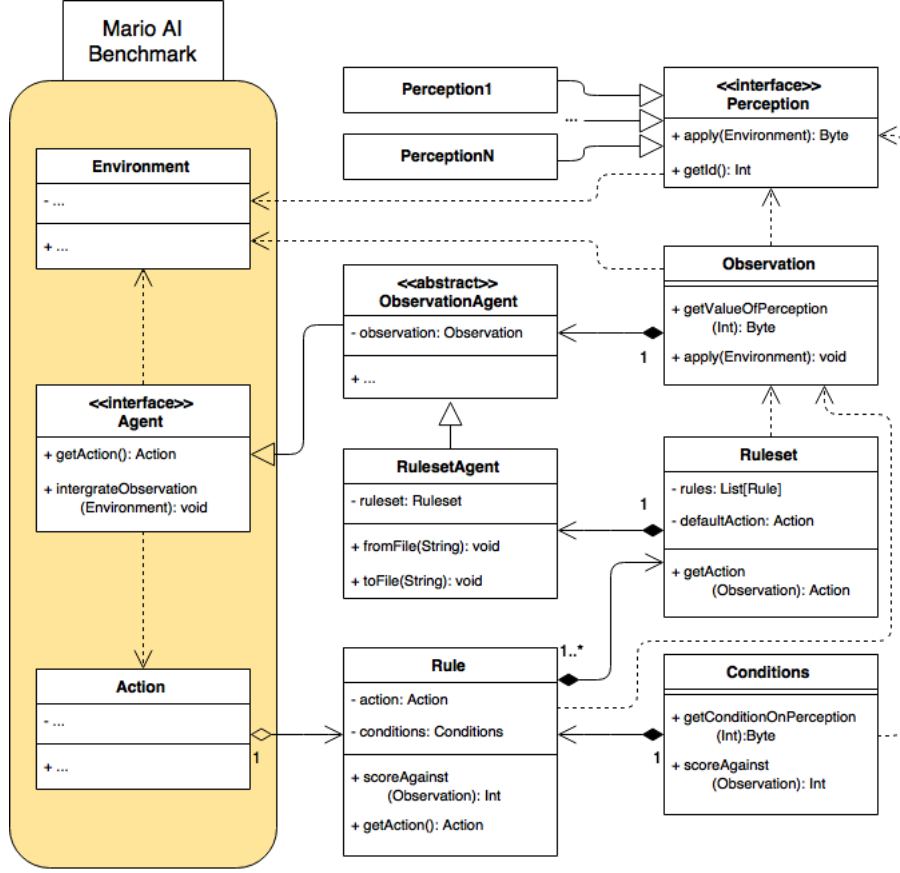


Figure 3: UML class diagram of the agent framework.

## 4 Agent Framework

### 4.1 Design

The benchmark offers an *Agent* interface, which when extended, can be passed into the game playing portion of the software. Every game tick the current *Environment* is passed to the agent, and then the agent is asked for an *Action*, which is a set of explicit key press for Mario to perform.

The agent framework’s basis will extend this interface. Similar to REALM’s agent, it will be a rule-based system. A UML class diagram of the system is given in Figure 3.

Each *Agent* instance is initialised with a ruleset, consisting of a list of rules, each containing an *Action*. On receiving the current game *Environment*, it is passed to a number of perception objects. Each one is responsible for a exactly one measurement (e.g. Enemy ahead, Mario moving to right etc.). These measurements are collected in an observation object and stored



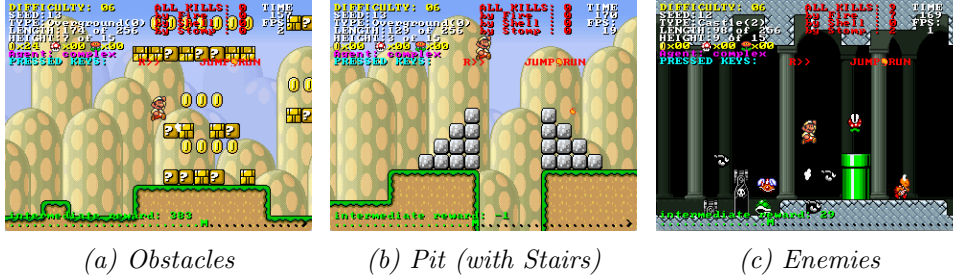


Figure 4: Example level features that agents must overcome.

in the agent.

When asked for an action, the *Agent* passes this observation to its ruleset, which tests it against each rule. Rules contain conditions on what perception measurements should be, which determine its score for a given observation. The highest scoring rule (with a conflict strategy for ties) is then asked for its action, which is returned from the agent.

Examples of the typical level features that the agent must detect and overcome can be seen in Figure 4.

Perceptions are hard coded; as such, all of an agents behaviour is determined by its ruleset. Thus, an implementation of the ruleset which allows persistence ensures that agents can be read and written to external files. Furthermore, rulesets are immutable and only information received during the current game tick is considered. Hence, agents have no state and are deterministic and thread safe (assuming the game engine is).

The capability of this framework is closely tied to the choice of perceptions. The more detail an agent is able to observe in its environment, the more potentially capable the agent can be. However, increasing the number of perceptions, and the detail they perceive at, increases the search space, making capable agents less likely to result from random mutation. Therefore, selection of perceptions must be carefully considered, with a balance being struck between detail and brevity.

The perceptions chosen were based upon those used in the REALM v1 agent [35, p. 85]. The following is the final list of perceptions chosen:

**MarioMode** Measures whether Mario is **small**, **big** or **fire** (capable of throwing fireballs).

**JumpAvailable** Detects if Mario has the ability to jump (i.e. on the ground or gripping a wall with the jump key off).

**OnGround** Measures whether Mario is in the air or not.

**EnemyLeft** Detects if an enemy is to the left of Mario, as seen in Figure 4c.

**EnemyUpperRight** Detects if an enemy is to the upper right of Mario, as seen in Figure 4c.

**EnemyLowerRight** Detects if an enemy is to the lower right or directly to the right of Mario, as seen in Figure 4c.

**ObstacleAhead** Detects if there is a block or terrain directly to the right of Mario, as seen in Figure 4a.

**PitAhead** Detects if there is a pit (falling into pits will end the level) to the right of Mario, as seen in Figure 4b.

**PitBelow** Detects if there is a pit directly below Mario, as seen in Figure 4b.

**MovingX** Measures the direction Mario is moving horizontally.

**MovingY** Measures the direction Mario is moving vertically.

The set of actions Mario can take is any combination of 6 key presses: **left**, **right**, **up**, **down**, **jump** and **speed** (which also shoots a fireball if possible). It was decided that this set would be restricted for use in the ruleset agent, only allowing combinations of **left**, **right**, **jump** and **speed**. This was to reduce the size of the rules, which reduces the search space of any learning process. The **up** key performed no function and the **down** key made Mario crouch, which seemed too niche to be useful during the design process.

## 4.2 Language and Tools

### 4.2.1 Scala

The dependency on the Mario AI Benchmark restricts the language choice to those that can run on the JVM. Therefore, Scala was chosen to be the primary language of the project. Its functional programming elements (pattern matching, list processing etc.) are very applicable to the ruleset and its semantic reasoner. Scala’s focus on immutability aids in maintaining the thread safety requirement. Furthermore, ECJ’s structure necessitates the use of type casting, which Scala handles elegantly. Several other Scala features were used throughout the project, such as lambdas, singletons, type aliases and case classes.

### 4.2.2 Maven

With two major and several minor dependencies, their management is important to the project. Maven was chosen to maintain this, as well as package the main project and both the sub-projects. It was chosen for its

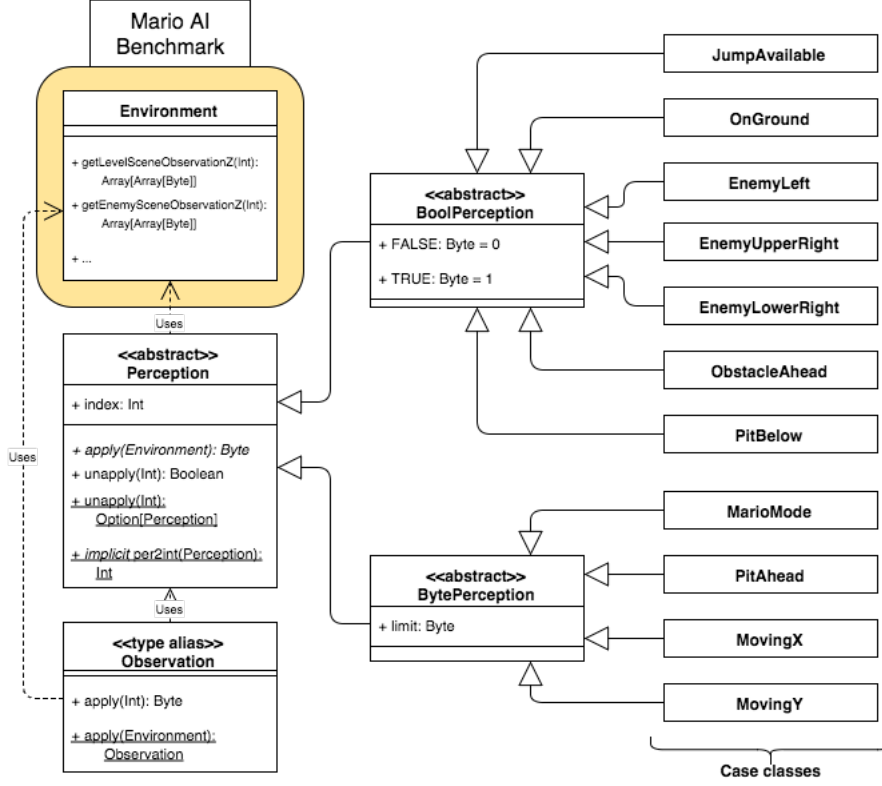


Figure 5: UML class diagram of the perception system.

ability to package both Java and Scala projects, keeping the tools consistent across the entire code base.

### 4.3 Implementation

The design of the agent framework allows for agents to be fully determined by their ruleset. In order for an agent to be used in an evolutionary algorithm, it must be represented by a simple data structure. Hence, implementation must allow for agent’s rulesets to be unambiguously represented as, for example, a one dimensional collection. This section will describe the implementation steps taken to represent rulesets as a array of 8-bit integers, utilising Scala’s built in types Vector and Byte.

#### 4.3.1 Perceptions

The Perception interface was implemented as an abstract class, with an index integer field and an apply method. *apply* takes the game Environment and returns a Byte (an 8-bit integer) instance, representing the measurement. Perception was extended into two further abstract classes: BoolPer-

ception, which enforces `apply`'s return be either 1 or 0, representing true and false respectively; and `BytePerception`, which has a `limit` field and enforces `apply`'s return is between 0 and `limit` (inclusively). A full list of perceptions and what their byte values represent can be found in Table A1 in Appendix A.

Concrete perceptions were implemented as objects (singletons) using the `case` keyword, which allows for exhaustive pattern matching on the `Perception` class, as well as type safe extension by adding further case objects. Each one declares a unique index integer to the `Perception` superclass (starting at 0 and increasing by one for each). They implement the `apply` method to return their specific measurement. Figure 5 shows a class diagram of the this system.

For illustration, consider these examples. The perception `PitBelow` extends `BoolPerception`, with a unique index of 8 and implements `apply` to return 1 if there is a pit directly below Mario and 0 otherwise. `MarioMode` extends `BytePerception`, with a unique index of 0 and a limit of 2, with `apply` returning 0 if Mario is **small**, 1 for **big** and 2 for **fire**.

This approach keeps all the information about perceptions contained in one file. The number of perceptions is easily extended by adding more case objects. Furthermore, it allows the `Observation` and `Conditions` classes to be implemented as fixed length byte vectors, with the vector's index matching the perception's unique index field. With use of Scala's implicit and extractor functionality, building the `Observation` and validating the `Conditions` vectors is type safe and concise:

```
val observationVector =
  Vector.tabulate(Perception.NUMBER_OF_PERCEPTIONS) {
    n: Int => n match {
      // This retrieves the perception object with the index
      // that matches n.
      case Perception(perception) => perception(environment)
    }
  }

def validateConditions(conditionsVector: Vector[Byte]): Boolean = {
  conditionsVector.zipWithIndex.forall {
    case (b: Byte, Perception(perception)) => perception match {
      case boolP : BoolPerception =>
        (b == boolP.TRUE) || (b == boolP.FALSE)
        || (b == DONT_CARE)
      case byteP : BytePerception =>
        ((0 <= b) && (b <= byteP.limit)) || (b == DONT_CARE)
    }
    case _ => false
  }
}
```

Notice that no information about specific concrete perceptions is required, enforcing the open-closed design principle and allowing perceptions

to be added without need to alter this code.

### 4.3.2 Perceiving the Environment

The *Environment* interface contains several individual methods that report Mario's situation. Therefore, implementing Perceptions that concerned Mario (e.g. MarioMode, MovingX etc.) was trivial.

Perceptions pertaining to enemies, obstacles or pits was more challenging. Environment provides two 19x19 arrays, one for enemies and one for terrain. Centred around Mario, each array element represents a 'square' (16 pixels) of the level scene. The value of an element marks the presence of an enemy or terrain square.

Enemy and obstacle perceptions pass the relevant array, a lambda test function, coordinates for a box segment of the array and a boolean to a helper function. This function uses Scala's for loop comprehensions to search through the box, applying the lambda to each element, returning the boolean parameter if the lambda returns true at least once. In this way it is easy to search for an enemy or obstacle in a box relative to Mario. Pits work in a similar way, but declare columns instead (see Listing A1 in Appendix A). If there is no terrain in the column below Mario's height, then it is considered a pit. Take EnemyLeft for example:

```
case object EnemyLeft extends BoolPerception(3) {
  // Minus = (Up,Left) | Plus = (Down,Right)
  val AREA_UL = (-2,-2); val AREA_BR = (1, -1);

  def apply(environment: Environment): Byte = {
    if(Perception.enemyInBoxRelativeToMario(environment,
      AREA_UL, AREA_BR)) 1 else 0
  }
}

def enemyInBoxRelativeToMario(environment: Environment,
  a: (Int, Int), b: (Int, Int)): Boolean = {
  val enemies = environment.getEnemiesObservationZ(2);
  val test = (grid: Array[Array[Byte]], tup: Tuple2[Int, Int]) =>{
    val x = grid(tup._1)(tup._2)
    x == 1
  }
  checkBox(enemies, test, getMarioPos(environment), a, b, true)
}
```

```

def checkBox(grid: Array[Array[Byte]],
             test: (Array[Array[Byte]], (Int, Int))=>Boolean,
             mario: (Int, Int),
             a: (Int, Int), b: (Int, Int),
             ret: Boolean): Boolean = {

  val relARow = min(grid.length-1, max(0, (a._1 + mario._1)))
  val relACol = min(grid(0).length-1, max(0, (a._2 + mario._2)))
  val relBRow = min(grid.length-1, max(0, (b._1 + mario._1)))
  val relBCol = min(grid(0).length-1, max(0, (b._2 + mario._2)))

  for {
    i <- min(relARow, relBRow) to max(relARow, relBRow)
    j <- min(relACol, relBCol) to max(relARow, relBCol)
    if (test(grid, (i, j)))
  }{ return ret }
  !ret
}

```

### 4.3.3 Observation, Conditions and Actions

For clarity an Action class was created for use in the ruleset, with an adapter method to convert it into the boolean array expected in the Agent interface. As the Observation and Conditions classes were to be implemented as fixed length byte vectors, so was the Action class.

Action vectors have a fixed length of 4, where elements represent **left**, **right**, **jump** and **speed** respectively. Observation vectors have a fixed length equal to the number of perceptions and hold the byte returned by each Perception's apply function. Conditions have the same length and hold data in the same range as Observation, the condition on each perception therefore being that the corresponding element in the observation be equal. They have one additional possible value, DONT\_CARE (equal to byte -1), which represents that no condition be placed on that perception.

Instead of implementing these classes as wrappers for *Vector[Byte]*, which can be inefficient and overly verbose, type aliases were used. This allowed each class to be referred to explicitly (rather than just by variable name), which provides readability and type safety, whilst still having direct access to the list processing methods included in the Vector class. They were declared on the agent framework's package object, making them accessible package wide. An object with static data and factory methods was included for each.

For example, this allowed Observation to be used as such:

```

abstract class ObservationAgent extends Agent {
  // Using factory method for a blank observation
  var observation: Observation = Observation.BLANK
  ...
}

```

```

def integrateObservation(env: Environment): Unit = {
  // Using the Observation factory method
  // to build a new observation
  observation = Observation(env)
  if (printObservation) {
    // Using Vector method foreach directly
    observation.foreach {
      b:Byte => print(" ~ " + b)
    }
  }
  ...
}

```

#### 4.3.4 Rules and Rulesets

Having both Conditions and Action implemented as byte vectors allows rules to be represented in the same way. Each rule is simple the concatenation of the Conditions and Action vectors. The rule vector are fixed length as both Conditions and Action are. Moreover, as rulesets contain just a list of rules, rulesets can be unambiguously represented by a single dimension byte vector. This allows rulesets not only to be persisted easily (as say a csv) but also gives the data representation needed for the evolutionary process.

In this case both Rule and Ruleset were implemented as wrapper classes for *Vector[Byte]* and *Seq[Rule]* respectively. Ruleset also holds a default Action, which is used if no rule matches the environment.

The semantic reasoner of the rule system is split across both classes. In Rule, the *scoreAgainst* method is passed the observation and produces a score by looping through the conditions and adding 1 if the condition and observation match in value and 0 if the condition is DONT\_CARE. If a mismatched condition is found, the method immediately returns with -1. It is implemented tail recursively to provide maximum efficiency.

```

def scoreAgainst(observation: Observation): Int = {
  val conditions = ruleVector.slice(0, Conditions.LENGTH)
  @tailrec
  def scoreRecu(i: Int, sum: Int = 0): Int = {
    if (i == Conditions.LENGTH) sum
    else conditions(i) match {
      case Conditions.DONT_CARE => scoreRecu(i+1, sum)
      case b if b == observation(i) => scoreRecu(i+1, sum+1)
      case _ => -1
    }
  }
  scoreRecu(0)
}

```

In Ruleset, the *getBestAction* is passed the observation and returns an action boolean array. Using tail recursion it performs a fold operation on

its rules, saving and returning the best scoring rule (preferring earlier rules when tied). If no rule gets a score of 0 or above then the default action is returned.

```
def getBestExAction(observation: Observation): Array[Boolean] = {

  @tailrec
  def getBestRuleRecu(rs: Seq[Rule], best: Option[Rule] = None,
    bestScore: Int = 0): Option[Rule] =
    rs match {
      case Nil => best
      case (r +: ts) => {
        val newScore = r.scoreAgainst(observation)
        if (newRuleBetter(bestScore, newScore))
          getBestRuleRecu(ts, Some(r), newScore)
        else
          getBestRuleRecu(ts, best, bestScore)
      }
    }

  getBestRuleRecu(rules) match {
    case None => defaultAction.toBooleanArray
    case Some(r) => r.getAction.toBooleanArray
  }
}
```

#### 4.3.5 Persistence

Agent's are persisted by persisting their ruleset. Rulesets are persisted in single line csv files. An IO helper object is passed an agent, extracts its ruleset and requests its vector representation, writing each byte separated by a comma. On reading an agent file, it constructs a byte vector. This byte vector is passed to the Ruleset's factory method, which groups the vector by rule length to form the rule sequence.

### 4.4 Testing

Due to the agent modules heavy reliance of the Environment interface the use of a mocking facility was required. The ScalaMock testing library was adding to the project to provide this.

Perceptions were unit tested individually, using white-box approach (due to the inclusion of mocking). Each test stubbed the Environment interface, instructing it to return a specific value (or array) for the relevant call, and testing that the perception echoed or processed it correctly. This allowed Perceptions to be tested independently of the game engine and provided test coverage. However, as there was very little documentation Environment interface, expected return values had to be investigated manually and edge cases could have easily been missed.



Rulesets (and Rules) were tested with a largely black-box end-to-end style. This was required due to the reliance of type aliases. Fixed rulesets were constructed to verify that the *getAction* method returned the expected action based on a fixed observation. Mocking of individual rules was not used in case the Rule class was altered to be a type alias instead of a wrapper class.

These tests were added to Maven’s build lifecycle, and hence run on every project build.

## 4.5 Handcrafted Agents

For the purpose of evaluation and comparison three handcrafted rulesets were created for the agent framework. Full rulesets for these agents can be found in Appendix B.

**Forward Jumping** This ruleset commands the agent to jump whenever it can, regardless of its surroundings. It contains a single rule and a default action. It is a blind agent that does not take advantage of the framework, however it is surprising effective. An analogous agent was used for comparisons in the 2009 Mario AI Competition and was found to score higher than many entrants [26, p. 7]. The learning process will aim to discourage this behaviour as it is neither interesting nor optimal.

**Simple Reactive** This agent only jumps when it detects an enemy, obstacle or pit in its way. It contains 5 rules and defaults to moving right at speed. This agent makes better use of the framework, but still does not use all of its perceptions. Its behaviour is more interesting, however it tends to score similarly to the forward jumping agent. Despite low attainment, a learnt agent that behaves similarly will be evaluating more favourably, because it is making use of the agent’s perceptions.

**Complex** This agent is the most interesting and highest scoring of the three. It has a multitude of behaviours and builds off of the simple reactive agent. It contains 18 rules and makes use of all perceptions except MarioMode and EnemyLeft. Its behaviour was investigated at length and as such will form a good comparison to the final learnt agent. Effective evolved behaviours unconsidered in this agent’s creation are a sign of the validity of the learning process.

Creation of the handcrafted rulesets also informed additions and alterations to the agent’s perceptions. For example, it was originally difficult to create an effective pit strategy and so PitAhead was changed from a BoolPerception to a BytePerception, returning 3 values representing **none**, **far** and **close**.

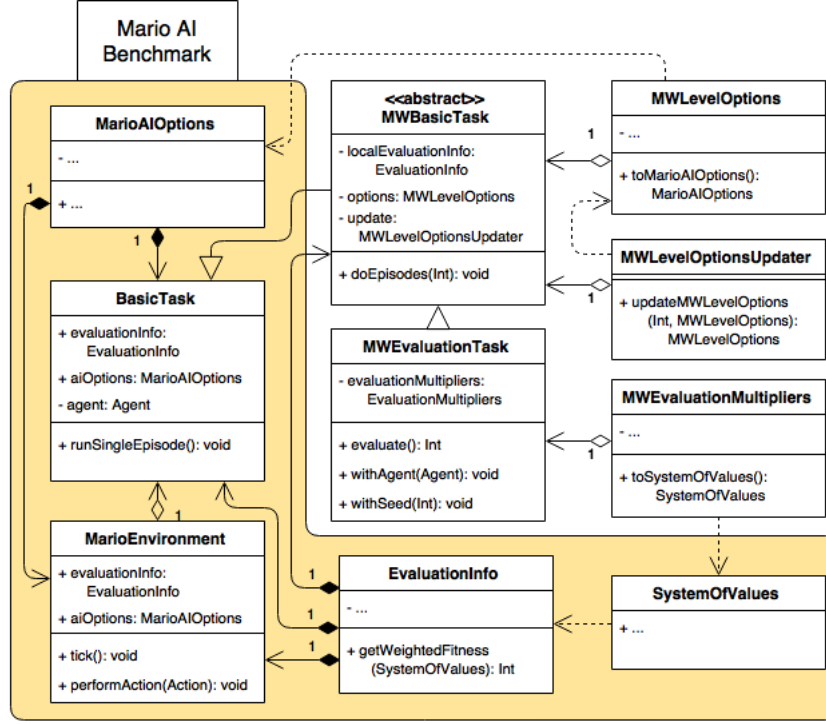


Figure 6: UML class diagram of the level playing module.

## 5 Level Playing Module

The main purpose of the level playing module is to evaluate the fitness of agents during the learning process. An effective learning process needs a diverse testbed, thus the level playing module must be deterministic, highly configurable and able to provide variety.

### 5.1 Design

The heart of the game engine is the benchmark’s *MarioEnvironment* class. It is responsible for calling the *LevelGeneration* class, updating the scene each tick with an action, and reporting the scene through the Environment interface. Parameters controlling level generation and playing are contained in the *MarioAIOptions* class. The *BasicTask* class controls the game loop in conjunction with an agent. It initialises the *MarioEnvironment* class with options, then runs through the loop, commanding a tick, passing the environment to the agent and finally requesting an action and passing it to the *MarioEnvironment* instance. Statistics pertaining to the agent’s performance are stored in *MarioEnvironment* in the *EvaluationInfo* class, which are cloned to *BasicTask* at the end of the level. Fitness can be requested from *EvaluationInfo* with the optional parameter, a *SystemOfValues* instance,

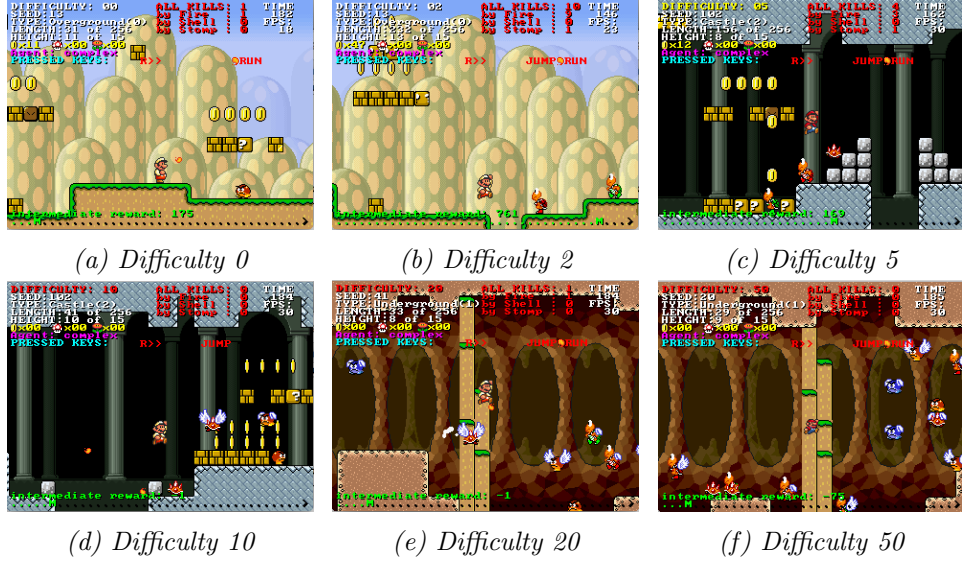


Figure 7: Handcrafted **complex** agent playing on different difficulty levels.

which contains the multipliers for various measurements.

The MarioAIOptions class contains several useful parameters. They include: an integer seed, which is passed to the level generator’s RNGs; a boolean that toggles visualisation; integers that determine level length and time-limit; and booleans that toggle enemies, pits and blocks. One parameter of particular importance is level difficulty, which has an effect on nearly all factors of level generation. Examples of different difficulties can be found in Figure 7.

The SystemOfValues class contains multipliers for distance achieved, whether or not the level was completed, the amount of time left on completion as well as many others.

The level playing module will extend this system. Its primary objective will be to allow for multiple levels (episodes) to be played with options that update as prescribed by some parameter class. Upon completion it will produce a fitness based on all levels played. Furthermore, it will allow for the injection of different agents and level seeds, to allow different agents to play the same options without rebuilding the instance. A UML class diagram for the module can be found in Figure 6.

Level options are stored in the *MWLevelOptions* class, which acts as an adapter for the MarioAIOptions class. Each new level these will be updated by the dedicated *MWLevelOptionsUpdater* class. The *MWEvaluationMultipliers* class is an adapter for the SystemOfValues class, which is used to calculate the fitness at the end of a sequence of levels. These classes are separated from the game playing task and are included using constructor

dependency injection. This helps to decouple the system and improve the ability to modify and test.

Both `MWLevelOptions` and `MWEvaluationMultipliers` are designed to be data classes, which provides determinism and thread safety, as well as easy persistence. `MWLevelOptionsUpdater`'s qualities in this regard are Implementation details and are discussed in a following section. A full list of field for both the `MWLevelOptions` and `MWEvaluationMultipliers` classes can be found in Appendix C.

## 5.2 Implementation

### 5.2.1 Parameter Classes

The `MWLevelOptions` and `MWEvaluationMultipliers` classes were implemented as data classes in an immutable builder pattern style. Each field has a *withField(field)* method that returns a cloned instance with that field changed. This affords a concise, declarative style, whilst maintaining immutability. `MWEvaluationMultipliers` has a implicit converter to a `SystemOfValues` instance, which is required for evaluation in the benchmark. However, a similar approach was not possible for converting `MWLevelOptions` to a `MarioAIOptions` instance (required for the game-engine). `MarioAIOptions` hold more parameters than `MWLevelOptions` (e.g. agent and level seed), therefore the adapter function takes both the current `MarioAIOptions` and a `MWLevelOptions` instance as parameters and updates the former by overriding the corresponding fields with values from the latter.

`MWLevelOptions` was not implemented as a class. Instead, `MWBasicTask` expects a lambda function. This lambda takes the episode number and the current `MWLevelOptions`, returning an updated set of options. `MWLevelOptions` builder structure ensures this is always a new instance, and hence maintains immutability. The inclusion of the episode number allows the function to remain deterministic. Moreover, with the ability to build closures in Scala, this lambda can be built from data (for example, a list of options, where indexes relate to episode number).

### 5.2.2 Persistence

The primary use of the level playing module is during the evaluation stage of the learning process. Hence, it was decided to use ECJ's parameter file system to persist level playing parameters, which allows them to be written in the same file as the rest of the learning parameters.

ECJ's parameter system builds upon Java's Properties file system. From a parameter file (formatted in the Java Properties format) a `ParameterDatabase` can be built, from which specific parameters can be requested

using the `Parameter` class. This system was used to persist the two parameter classes, `MWLevelOptions` and `MWEvaluationMultipliers`; the level options update lambda data; and other level playing data such as number of levels (episodes) and base level seed. For example, the following lines would set the number of levels to 10 and the base difficulty to 5:

```
level.num-levels = 10
level.base.difficulty-num = 5
```

A static utility class `EvaluationParamsUtil` was created to handle the reading of the level playing data from these files. A `ParameterDatabase` is built and passed to utility class, which builds the required parameter class. For `MWLevelOptions` and `MWEvaluationMultipliers` it searches for the prefixes `'level.base'` and `'mult'` respectively and then looks for suffixes corresponding to specific fields. If a field's suffix is not found then it is initialised to a default value (which is always zero for `MWEvaluationMutlipliers`).

The update lambda is built as a closure on a collection of `Map` instances, one for each field in `MWLevelOptions`. For each `n` from 0 to the number of levels (episodes), the utility function looks for the prefix `'level.{n}'`, which is used to hold the update for episode `n`. For each field a `Map` is built and using the same suffixes as for `MWLevelOptions`, key-value pairs are added mapping `n` to the value found. When the update lambda is called, it consults these maps and updates the `MWLevelOptions` with the new value if one is found for the current episode number.

For example, if the parameter file contained the following lines:

```
level.num-levels = 4
level.base.enemies = false
level.1.enemies = true
level.3.enemies = false
```

Then enemies would be off for the first episode, on for the second and third and off again for the fourth and final episode.

### 5.2.3 The Episode Loop

The entry point of the level playing module is the `MWEvaluationTask` class, which extends the abstract `MWBasicTask` class. `MWEvaluationTask` is instantiated with a base set of options (as `MWLevelOptions`), a `MWEvaluationMutlipliers` instance and an update lambda, as well as the number of episodes (levels) to run. The agent and base level seed can be injected with the *`withAgent(agent)`* and *`withLevelSeed(seed)`* methods (which also reset of evaluation information).

```

class MWEvaluationTask(val numberOfLevels: Int,
                      val evalValues: MWEvaluationMultipliers,
                      override val baseLevelOptions: MWLevelOptions,
                      override val updateOptionsFunc: (Int,
                                                         MWLevelOptions) => MWLevelOptions)
    extends MWBasicTask("MWMainPlayTask",
                        baseLevelOptions, updateOptionsFunc,
                        visualisation, args) with EvaluationTask
{

    private var baseLevelSeed: Int = 0;

    override def nextLevelSeed(episode: Int, lastSeed: Int) = {
        (3*episode) + lastSeed
    }

    override def evaluate: Int = {
        doEpisodes
        localEvaluationInfo.computeWeightedFitness(evalValues)
    }

    override def withAgent(agent: Agent): MWEvaluationTask = {
        super.injectAgent(agent, true)
        this
    }

    override def withLevelSeed(seed: Int): MWEvaluationTask = {
        baseLevelSeed = seed
        super.injectLevelSeed(seed, true)
        this
    }
}

```

On calling the *evaluate()* method, the number of levels is passed to the *doEpisodes(numberOfEpisode)* (a superclass method), which loops as follows:

```

def doEpisodes(amount: Int): Unit = {
    @tailrec
    def runSingle(iteration: Int, prevOptions: MWLevelOptions,
                  disqualifications: Int): Int = {
        if (iteration == amount) {
            disqualifications
        } else {
            // Calls the update lambda to get episodes set of options
            val newOptions = updateOptions(iteration, prevOptions)

            // Converts options to class required for game-engine
            val marioAIOptions = MWLevelOptions.updateMarioAIOptions(
                super.options, newOptions)
        }
    }
}

```

```

// Updates the level seed (which is set to increase each
// episode by MWEvaluationClass)
marioAIOptions.setLevelRandSeed(nextLevelSeed(iteration,
    marioAIOptions.getLevelRandSeed))

// Resets the evaluation information in the super class
super.setOptionsAndReset(marioAIOptions)

// Generates and runs the level using the super class
// Returns true if the agent was not disqualified
val notDisqualified: Boolean = runSingleEpisode(1)
val disqualification = if (!notDisqualified) 1 else 0

// Update the evaluation information for the entire run
// (as the super classes evaluationInfo gets reset every
// level
updateLocalEvaluationInfo(super.getEvaluationInfo)

// Loop
runSingle(iteration+1, newOptions, disqualifications +
    disqualified)
    }
}

// Sets the base options
super.setOptionsAndReset(MWLevelOptions.updateMarioAIOptions(
    options, baseLevelOptions))
disqualifications = runSingle(0, baseLevelOptions, 0)
}

```

Before every episode the options are updated by the lambda, updating the base options in the first episode. A new level seed is also requested from the MWEvaluationClass, which simply increases it each episode. These updates are converted and added to a MarioAIOptions instance and passed to the superclass in the benchmark. A single level is then generated and played using the superclass. Evaluation information is added to MWBasicTask's local evaluation information (as the former is reset every episode) and the function loops tail recursively.

The function exits when the it has looped for each level. At this point the *evaluate()* method requests a fitness from the local evaluation information, passing in the evaluation multipliers, which is then returned.

### 5.3 Modifying the Benchmark

Preliminary runs of the benchmark software revealed several issues and defects. As the learning algorithm would run over several generations, any errors could halt it prematurely, which could be costly in terms of project time keeping. In order to address this a Java project was created from the benchmark's source code and fixes were made. This code was packaged with Maven and included as a dependency in the main project.

Several minor exceptions were caught or addressed, however the two largest issues failed ‘quietly’. They concerned the LevelGeneration class and surrounded enemy and pit generation with regards to level difficulty. Fixes were made to ensure level difficulty scaled more consistently.

### 5.3.1 Enemy Generation

In observing the benchmark generated levels it was apparent that enemy density was very high, even on lowest difficulties. Examining the LevelGeneration class revealed this to the result of what was probably unintended behaviour.

Levels are generated in zones of varying length. Quite often, a zero length zone is created, which has no effect on terrain. However, enemies were still being added to these zones, creating very high density columns of enemies during levels. This was addressed, with the addition of a more gentle enemy count curve and better spacing.

### 5.3.2 Pit Generation

Another apparent shortcoming was pit length. Pits were only of two sizes, small and very large, and after a certain level difficulty were always very large. Although this was intended behaviour, comments from the original developers suggest it was a placeholder for a more sophisticated system. An edit was made to scale maximum pit length on level difficulty. Each pit’s length is chosen probabilistically on a bell curve, which is shifted by level difficulty.

## 5.4 Testing

The benchmark’s BasicTask class is tightly coupled to the MarioEnvironment, which means that we were unable to mock the game-engine for testing. This problem extends to the MWEvaluationTask class.

However, the decoupling of the parameter classes from MWEvaluationTask means that the persistence section of the module can easily be tested. The EvaluationParamUtil class is white-box unit tested by stubbing the ParameterDatabase interface and verifying the contents of the parameter classes requested. For example:

```
"getEvaluationMutlipliers" should "return eval mults from database
and zero otherwise" in {
  val base = blankParam.push(EvaluationParamsUtil.P_EVAL_BASE)
  (pdStub.exists _) when(base, *) returns(true)

  (pdStub.exists _) when(base.push(EvaluationParamsUtil.P_COINS),
    *) returns(true)
  (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil
    .P_COINS), *, *) returns(10)
```



Agent	Total Score	Levels Completed	Enemies Killed	Distance
<b>Complex</b>	1,817,195	171 (33%)	1498 (8%)	63,894 (48%)
<b>Simple Reactive</b>	1,095,287	88 (17%)	590 (3%)	43,286 (32%)
<b>Forward Jumping</b>	954,640	76 (15%)	677 (3%)	36,980 (28%)

Table 3: Competitive statistics from handcrafted agents playing the evaluation task with a seed of 10.

```

(pdStub.exists _) when(base.push(EvaluationParamsUtil.
  P_KILLED_BY_SHELL), *) returns(true)
(pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil
  .P_KILLED_BY_SHELL), *, *) returns(200)

(pdStub.exists _) when(base.push(EvaluationParamsUtil.P_DISTANCE
  ), *) returns(true)
(pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil
  .P_DISTANCE), *, *) returns(1)

val evalMults = EvaluationParamsUtil.getEvaluationMultipliers(
  pdStub, base);

assert(evalMults.coins == 10)
assert(evalMults.killedByShell == 200)
assert(evalMults.distance == 1)
assert(evalMults.win == 0)
assert(evalMults.kills == 0)
...
}

```

## 5.5 Comparator Task

In order to quantifiably compare agents, a competitive set of evaluation task options was created, modelled on those used during the final evaluation stage of the 2010 Mario AI Competition.

Agents play 512 levels, spread equally over 16 difficulty levels (0 to 15). Options such as enemies, pits, blocks etc. are periodically turned off for a level. Length varies greatly from level to level, with the time-limit being adjusted accordingly. Evaluation multipliers reward all possible positive statistics, such as enemy killed, coins collected and distance travelled. A full view of the comparator task’s parameter classes can be found in Appendix D.

The scores and other statistics attained by the three handcrafted agents playing the evaluation task, on seed 10, can be found in Table 3.

## 6 Learning

The learning module’s goal is to utilise the ECJ library to evolve a population of byte vectors (representing agent rulesets). This is achieved through extending ECJ’s classes, in conjunction with a parameter file. The ECJ library is an extensive collection of learning processes and, due to the limitations of this report, only the relevant sections, and the extensions thereof, are described. An important part of the project was adjusting the parameter file in reaction to previous learning runs, these revisions are discussed in Section 6.6. However, some parameters were deemed fundamental and stayed invariant throughout this process; these are described, alongside the general evolutionary approach, in the following section.

### 6.1 Evolutionary Approach

Similar to the approach taken in developing the REALM agent (described in Section 2.2.3), this project uses a  $(\mu + \lambda)$  evolutionary strategy. However, unlike REALM, it focuses purely on mutation, with no crossover between individuals (a technique not traditionally applied with ESes [41, p. ]).

Evolution runs for 1000 generations with a population size of 50. The truncation selection size for our ES ( $\mu$ ) is set to 5. Hence, the top 5 fittest individuals are chosen each generation. These are cloned and mutated to form 45 ( $\lambda$ ) new individuals, which are then joined by the original 5 for the next generation.

Each individual’s genome is a byte vector of length 300, representing a ruleset containing 20 rules. The default action is fixed over all rulesets and is specified by the parameter file.

Each element of the genome (a gene) has several mutation parameters associated to it. During the mutation phase each gene will be mutated with the probability prescribed by its mutation probability. Each gene can be mutating in one of two ways. The first will simply choose a random byte between its minimum and maximum values (inclusive), known as **reset** mutation. The second will favour one particular byte value, with a certain probability, which is known as **favour** mutation. Which method of mutation a gene undergoes, as well as the favoured byte and its probability are all contained in the individual gene’s parameters, allowing for very fine grained control over the mutation process.

The evaluation stage takes each individual’s genome and constructs an agent. The individual’s fitness is calculated by passing the agent to the MW-EvaluationTask, which has been initialised with a fixed set of level playing parameters decreed by the parameter file. These parameters have a custom set of evaluation multipliers and describe a task of 10 levels, varying level options for each level. Each generation builds this task with its own unique level seed to avoid over-fitting to one set of generated levels.

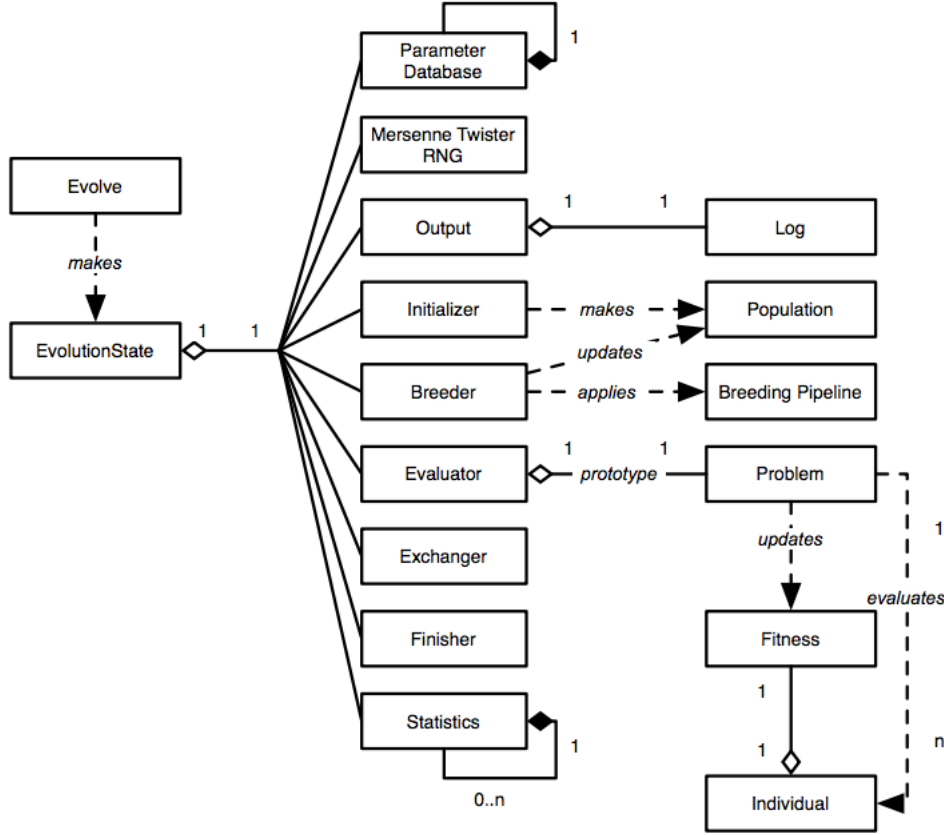


Figure 8: A top-level UML class diagram of the operation facilities in *EvolutionState*, taken from [40, p. 10].

Statistics for average and best fitness are logged each generation. Level seed is also reported, to allow direct comparison between the learning process and the hand-crafted agents. After the final generation a selection of agents are written to agent files, including the best individual of the final generation and the best individual over all generations.

## 6.2 The ECJ Library

The primary method for specialising ECJ for a particular use case is to extend the classes it provides, and specify these subclasses, alongside other pertinent values, in the parameter file. Nearly every method in ECJ's classes is passed the pseudo-global *EvolutionState* singleton, which holds instances of all other classes in use. A top-level view of ECJ's structure can be found in Figures 8 and 9.

The learning module will be built on top of the ECJ library, with a view to adhering to its style and ideology. For our approach, several of ECJ's

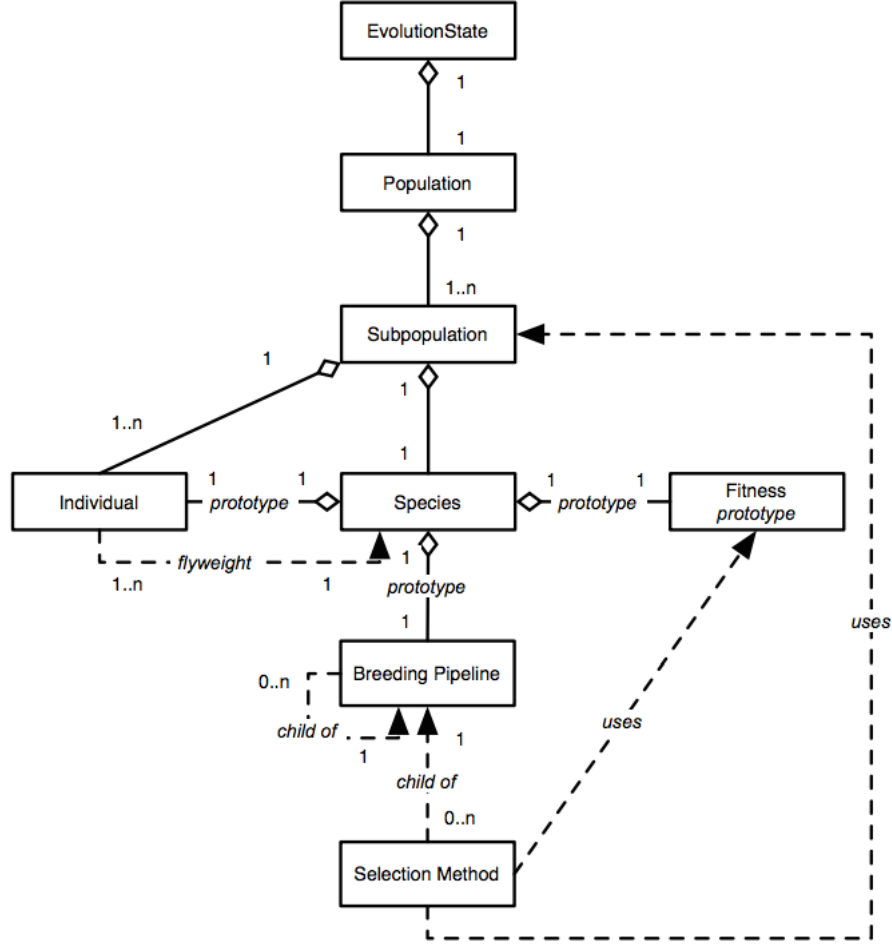


Figure 9: A top-level UML class diagram of ECJ's data objects, taken from [40, p. 11].

classes did not need to be extended, and simply specifying their default implementations in the parameter file was enough. The classes of interest to this project are the following: *Species*, *Individual*, *SelectionMethod*, *Breeder*, *BreedingPipeline*, *Evaluator*, *Problem* and *Statistics*. A simplified class diagram of the learning module is included as Figure 10.

ECJ provides the *ByteVectorIndividual* class, which contains a Java array of *bytes*, this is used to hold individual byte vector genomes. It also decreases the use of the *IntegerVectorSpecies* class, which holds the gene parameters (e.g. minimum value, mutation probability). Our design first extends this class to the *DynamicParameterIntegerVectorSpecies* class, which holds a subclass instance of another new class, the *DynamicSpeciesParameters* class. The specific subclass to be used is specified in the parameter file. The purpose of this class is to provide a facility to override the min, max and

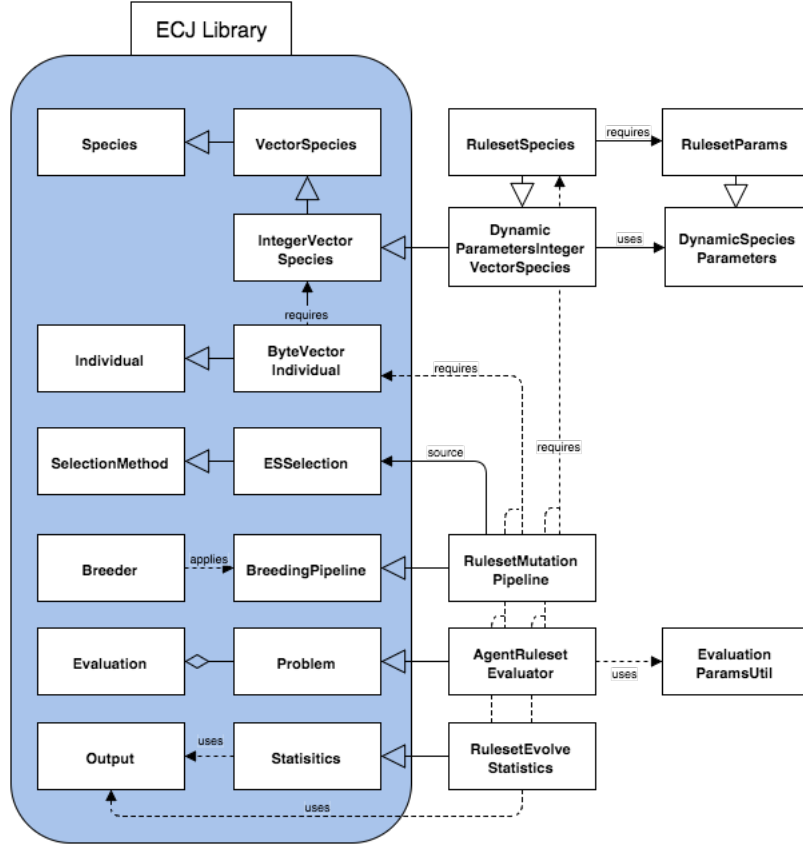


Figure 10: A simplified UML class diagram of the Learning module's extension to ECJ.

mutation probability properties of each gene directly from the agent framework (rather than the parameter file). This class is then further extended to the *RulesetSpecies* class, which holds gene parameters pertaining to favour mutation. This class also extends the parameter file's vocabulary to allow gene parameters to be specified on a **condition** or **action** (the composite parts of a rule) basis. For example, the favour byte for all genes that represent conditions can be set as follows in the parameter file:

```
...species.condition.favour_byte = -1
```

The *SelectionMethod* and *Breeder* classes are responsible for selecting individuals (based on fitness) to clone and pass to the breeding pipeline. ECJ natively supports evolution strategies and so it is enough to specify these classes to be ECJ's *ESSelection* and *MuPlusLambda* classes. The *BreedingPipeline* extension, *RulesetMutationPipeline*, performs the mutation of individuals. For each gene in an individual's genome it requests the corresponding mutation parameters from the species class. It then clones and, if required, mutates the gene with either favour or reset mutation.

The *Evaluator* class is responsible for threading and batching the individuals to have their fitness calculated by cloned instances of the *Problem* class. Hence, only the *Problem* class is extended, to the *AgentRulesetEvaluator* class. On initialisation this class reads the level options from the parameter file using the *EvaluationParamsUtil* class described in Section 5.2.2. Every generation, agents are constructed from the individuals of the population and passed, with the options and the level seed, to the *MWEvaluationTask* class. This returns a score, which is attached to the individual as its fitness.

Generational statistical reporting is handled by an extension of the *Statistics* class, which will output to log files specified in the parameter file.

### 6.3 Implementation

Extensions to the ECJ library were implemented in Scala, organised into a package and situated in the same project as the agent framework and level playing modules. This was to allow the level playing parameters to be read through ECJ's *ParameterDatabase* class (described in Section 5.2.2).

Nearly every class in the ECJ library implements a *setup(...)* method, which is used to initialise an instance. This method is passed the *EvolutionState* singleton, from which it can access the *ParameterDatabase*, which holds the contents of the parameter file. Classes can poll the *ParameterDatabase* by constructing a *Parameter* instance (in a composite pattern) with string keys.

#### 6.3.1 Modifying the ECJ Library

During implementation it became clear that the proposed species design was problematic due to the setup process of *IntegerVectorSpecies* and its superclass *VectorSpecies*. The setup method reads in values for gene minimum, maximum and mutation probability and then calls for the *Individual* prototype to be built. This means that any subclass of *IntegerVectorSpecies* cannot decide the values of the gene minimum, maximum and mutation probability. If they are written before calling setup on the superclass (which is required) they will be overridden, if they are written after the calling setup on the superclass they will not be built into the prototype. To address this the *VectorSpecies* class was modified to include the *prePrototypeSetup* method, which is called immediately before constructing the prototype in the setup method. Subclasses can extend this method to overwrite any properties set by *IntegerVectorSpecies* and *VectorSpecies*.

```

public void setup(final EvolutionState state, final Parameter base)
{
    // Add min, max, mutation probability etc.
    ...

    // Allow subclasses to override values
    prePrototypeSetup(state, base, def);
    state.output.exitIfErrors();

    // NOW call super.setup(...), which will in turn sets up the
    // prototypical individual
    super.setup(state, base);
}

// new method added
protected void prePrototypeSetup(final EvolutionState state, final
    Parameter base, final Parameter def) {
    //None by default, for subclasses
}

```

### 6.3.2 Species

IntegerVectorSpecies and VectorSpecies store gene parameters (e.g. min, max, mutation probability) as arrays of equal length to the genome, matching by index. These arrays are built during the setup method by polling the ParameterDatabase for the suffixes ‘min-value’, ‘max-value’ and ‘mutation-prob’. The parameter file can specify them globally, in segments or by individual gene (by index). For segments and single indexes the *loadParameterForGene* method is used, which takes the gene index and a prefix parameter (e.g. segment number or index). It builds a Parameter instance by appending the gene parameter suffixes to the passed prefix. It requests the Parameter’s value from the ParameterDatabase. If a value is found, it sets it as the element of the corresponding gene parameter array at the given index..

#### Dynamic Parameters

The DynamicParametersIntegerVectorSpecies performs the same task as described above, but obtains the values from a specified class rather than the parameter file. This class must be a subclass of the DynamicSpeciesParameters class. DynamicSpeciesParameter define methods for global, segment and index gene parameters, each return an Option on the value. The default implementation returns None, allowing subclasses to override only the methods they want to be used in the learning run. For our learning run we used the RulesetParams class, which implemented the *minGene(index)* and *maxGene(index)* methods. These methods return the min and max value for a particular gene, with the values being obtained via the agent framework:

```

def getIndexType(index: Int): IndexType = (index % ruleLength) match
{
  case n if n < conditionLength => Condition
  case _ => Action
}
override def maxGene(index: Int): Option[Int] = getIndexType(index)
match {
  case Action => Some(Math.max(MWAction.ACTION_FALSE, MWAction.
    ACTION_TRUE))
  case Condition => (index % ruleLength) match {
    case Perception(perception) => perception match {
      case bp : BoolPerception => Some(Math.max(bp.TRUE, bp.
        FALSE))
      case ip : BytePerception => Some(ip.limit.toInt)
    }
    ...
  }
}

```

The setup method of `DynamicParametersIntegerVectorSpecies` reads the subclass name (in this case `RulesetParams`) from the parameter file and instantiates it. The `prePrototypeSetup` method is then used to write the values (overriding those set by `IntegerVectorSpecies`). It calls each of `DynamicSpeciesParameters`' methods, if `None` is returned no action is taken, but if a value is return it is written to the parameter arrays.

```

override def prePrototypeSetup(state: EvolutionState, base:
  Parameter, default: Parameter): Unit = {
  if (dynamicParamsClassOpt.isDefined) {
    val dynamicParamsClass: DynamicSpeciesParameters
    = dynamicParamsClassOpt.get
    if (dynamicParamsClass.minGene.isDefined) {
      fill(minGene, dynamicParamsClass.minGene.get)
    }
    ...
    for(i <- 0 until genomeSize) {
      if (dynamicParamsClass.maxGene(i).isDefined) {
        maxGene(i) = dynamicParamsClass.maxGene(i).get
      }
      ...
    }
  }
  super.prePrototypeSetup(state, base, default)
}

```

## Ruleset Species

The `prePrototypeSetup` method of `RulesetSpecies` checks that the `DynamicSpeciesParameter` class used is `RulesetParams`, which contains several utility functions. Adding to globally, by segment and by index methods for declaring gene parameters, this setup also looks for Parameter prefixes 'condition'



and ‘action’. If found, it uses a utility function to run `loadParametersForGene` on all indexes corresponding to the prefix. This method is also passed the prefix, in this way `IntegerVectorSpecies` specific parameters such as min and max can be set on a condition or action basis without needing to implement them directly. For example, take the following line in the parameter file:

```
...species.condition.mutation-prob = 0.1
```

The ‘condition’ prefix is found by `RulesetSpecies` setup method, which indirectly loops through all indexes `x` that pertain to conditions:

```
override def prePrototypeSetup(state: EvolutionState, base:
  Parameter, default: Parameter): Unit = {
  super.prePrototypeSetup(state, base, default)
  ...
  if (state.parameters.exists(base.push(RulesetSpecies.P_CONDITION)
    , default.push(RulesetSpecies.P_CONDITION))) {
    dpc.runOnIndexes(Condition, genomeSize) {
      (x: Int, mod: Int) => {
        loadParametersForGene(state, x,
          base.push(RulesetSpecies.P_CONDITION),
          default.push(RulesetSpecies.P_CONDITION), "")
      }
    }
  }
}
```

It passes the condition prefix and the index to the `loadParameterForGene` method. In `IntegerVectorSpecies`, the suffix ‘mutation-probability’ is added to the condition prefix and the value 0.1 is found, and set for all indexes `x` in the mutation probability array.

`RulesetSpecies` also extends the `loadParametersForGene` method to read in the values for favour mutation from the parameter file. This is controlled by three arrays: `favourMutation`, a boolean array which holds whether or not a gene should be favour mutated; `favourByte`, a byte array holding the favoured byte; and `favourProbability`, the probability with which this byte will be chosen. By extending `loadParametersForGene` these parameters can be read in on condition or action bases, as well as by segment or individual index.

### 6.3.3 Mutation

To implement the mutation strategy, the `RulesetMutationPipeline` was created. It extends the `BreedingPipeline` and overrides the `produce` method. It makes no use of the setup method as all mutation parameters are stored in the `Species` class. The breeding pipeline is threaded, and as such `RulesetMutationPipeline` is prototypes and cloned for each breed thread. The `produce` method is called to mutate batches of `Individuals`, which vary in

size.

RulesetMutationPipelines produce method first calls produce on its source (in this case the ESSelection class), which fills the array of individuals to be mutated. It verifies that these individuals are ByteVectorIndividuals and their Species is RulesetSpecies. It then clones each individual and resets its fitness. Lastly it loops through each individual and passes it to the *mutateIndividual* method.

The mutateIndividual method loops through the individual's genome by index. It uses this index to acquire each gene's corresponding parameters from the species class (e.g. minimum, maximum, mutation probability, favour mutated, favour byte etc.). Using the thread's random number generator stored in the EvolutionState object it decides whether to mutate by calling `nextBoolean(mutationProbability)`. If so it mutates the gene by favour or reset mutation (again decided by the gene parameters). For reset mutation it replaces the gene with a random byte between the gene's minimum and maximum. For favour mutation it makes another call to `nextBoolean`, with the favour probability parameter. If this returns true, the gene is replaced by the favour byte, otherwise a byte between the minimum and maximum is chosen, excluding the favour byte. The code for this method is given below (edited for brevity).

```
protected def mutateIndividual(state: EvolutionState, thread: Int,
  vecInd: ByteVectorIndividual, species: RulesetSpecies):
  ByteVectorIndividual = {
    for (n <- 0 until vecInd.genome.length) {
      if (state.random(thread).nextBoolean(species.
        mutationProbability(n))) {
        if (species.favourMutation(n)) {
          if (random.nextBoolean(favourProbability))
            vecInd.genome(n) = species.favourByte(n)
          else
            vecInd.genome(n) = getRandomByte(
              species.minGene(n).toByte,
              species.maxGene(n).toByte,
              state.random(thread),
              species.favourByte(n)
            )
        } else {
          vecInd.genome(n) = getRandomByte(
            species.minGene(n).toByte,
            species.maxGene(n).toByte,
            state.random(thread)
          )
        }
      }
    }
    vecInd.evaluated = false
    vecInd
  }
```

### 6.3.4 Evaluation

The Evaluator class prototypes the AgentRulesetEvaluator class and holds one clone for each evaluation thread. The AgentRulesetEvaluator overrides three methods from the Problem class: *setup* which is called before creating the prototype, *prepareToEvaluate* which is called once per thread clone before evaluation, and *evaluate* which is called on single individuals multiple times per thread.

The setup method loads in the MWLevelOptions, MWEvaluationMultipliers, update lambda, number of levels, generational level seeds and the default ruleset action using the EvaluationParamUtil class. As these are built into the prototype they are shared across all evaluation threads, taking advantage of their immutability (explained in Section 5.2.1).

The MWEvaluationTask instance is built with these options once per thread, in the prepareToEvaluate method. As the game loop is mutating process it cannot be shared across threads, but as the task can be reset and new agents injected, it can be used multiple times in one thread.

The evaluate method is passed an individual, which is verified to be a ByteVectorIndividual. The individual's genome is built into a ruleset, which is used to initialise an MWRulesetAgent. This agent, along with the level seed for the current generation, is injecting into the task. The task then evaluates the agent, returning a score, which is attached to the individual as its fitness.

```
override def evaluate(state: EvolutionState, individual: Individual,
  subpop: Int, thread: Int): Unit = {
  individual match {
    case ind: ByteVectorIndividual => {
      if (task.isDefined) {
        val evalTask = task.get
        val name = this.buildIndAgentName(state, individual,
          subpop, thread)
        val ruleset: Ruleset = Ruleset.buildFromArray(ind.
          genome, defaultAction)
        val agent: Agent = MWRulesetAgent(name, ruleset)

        val iFitness = evalTask.withAgent(agent)
          .withLevelSeed(_taskSeeds(state.
            generation))
          .evaluate

        ind.fitness match {
          case _: SimpleFitness => {
            ind.fitness.asInstanceOf[SimpleFitness].
              setFitness(state, iFitness.toDouble, false
            )
            ind.evaluated = true
          }
        }
      }
    }
  }
}
```

```

        case _ => {
            state.output.fatal("This evaluator (
                                EvolvedAgentRulesetEvaluator) requires a
                                individuals to have SimpleFitness")
        }
    } else {
        state.output.fatal("Task was not defined when
                            evaluating individual, implying prepareToEvaluate
                            was not run on this instance.")
    }
}
case _ => {
    state.output.fatal("This evaluator (AgentRulesetEvaluator)
                        requires a ByteVectorIndividual")
}
}
}

```

### 6.3.5 Statistics

The Statistics class and its subclasses are setup at the beginning of the learning run and then called as ‘hook’ at various points during the generational loop. RulesetEvolveStatistics implements *setup* and two of the hooks, *postEvaluationStatistics* and *finalStatistics*. The setup method reads the filenames for the generation log, final log and the final agent exports. It registers the log file with EvolutionState’s *Output* instance to allow it to be called from the other methods.

The postEvaluationStatistics method is passed the EvolutionState, from which it can access the current population and generation number. It loops through the individuals of the current population, calculating the average fitness and determining the best individual. The average fitness, best fitness, current level seed, generation number and the best individual’s genome are all written to the generation log using the Output class:

```

val all = "~all~ " + genNum + "," + levelSeed + "," + avScore + ","
          + bestScore
state.output.println(
    "----- GENERATION " + genNum + " -----\\n", genLog)
state.output.println(all +
    "\\nLevel Seed : " + levelSeed +
    "\\nAverage Score : " + avScore +
    "\\nBest Score : " + bestScore +
    "\\nBest Agent :-" +
    "\\n " + agentStr +
    "\\n-----\\n\\n", genLog)

```

The best individual in the generation is saved to the currentBestIndividual field. Further checks to see if it is the best overall individual or has the biggest difference to the average are also performed, resulting in the in-

dividual being saved to the `bestOverallIndividual` and `biggestDiffIndividual` fields respectively. These checks are also controlled by generation number, only being performed after a certain generation, specified in the parameter file.

The `finalStatistics` method retrieves the `currentBestIndividual`, `overallBestIndividual` and `biggestDiffIndividual`. From each it constructs a ruleset, initialises an agent with the correct default action and passes it to agent IO utility class to be persisted into the corresponding filenames captured during setup. The entire final generation and their fitness values are also written to the final log.

## 6.4 Testing

ECJ's structure makes unit testing extremely difficult, due mainly to its heavy reliance on the `EvolutionState` singleton. It is utilised by every class and passed to nearly every method. `ScalaMock` is unable to mock this object as it is not accessed through an interface and building a fixed instance would be overly time consuming (and tantamount to simply running the software).

Ultimately, besides the `EvaluationParamsUtil` class, no testing was performed. Instead, the implementation adopted a 'fail loudly' approach, where any discrepancy was logged and triggered a program shutdown. In this way errors were easily noticed and corrected.

The lack of testing is not desirable, and given more time, efforts would have been made to rectify this. However, as the software has no external 'users' (it being used as a tool solely by those that wrote it) it did not affect the final result.

## 6.5 Running

As learning could take several hours, or even days, the software was run on an external quad-core server. Maven was used to package the main project, with all its dependencies in to a jar file, which allowed it to easily transferred and run. As the project went through several revisions and performed many learning runs, shell scripts were written to facilitate this process. The push script called Maven to package the project, running all tests, which was then copied to the server (using `ssh`), along with parameter files and their runner scripts. These runner scripts were then run remotely to begin the learning process. On completion another script retrieved the generation log and the agent files.

Immutability and thread safety were a major focus of the project. This was done to allow the evaluation and breeding stages to be run over four threads (to take advantage of the server's quad-core). However, during the initial run it was quickly discovered that multithreading the evaluation process was impossible. Even though each thread had its own evaluation

Level	Difficulty	Enemies	Type	Length	Time	Other
1	2	N	1	200	100	
2	3	N	0	200	100	
3	5	N	2	200	100	
4	10	N	0	200	100	
5	2	N	0	200	100	Flat Level
6	7	N	0	200	100	Flat Level
7	2	Y	0	200	100	Frozen Enemies
8	2	Y	0	200	100	
9	3	Y	1	200	100	Tubes
10	5	Y	0	200	100	Cannons, no Blocks

Table 4: Level playing parameters of the LEMMEL learning run. Type 0 is outside, and type 1 and 2 are inside.

Statistic	Multiplier
Distance	1
Completion	3200
Mario Mode	200
Enemy Kills	100
Time Left	2

Table 5: Evaluation multipliers of the LEMMEL learning run.

Gene Parameter	Probability
Condition Mutation	0.05
Condition Favour	0.5
MarioMode Favour	0.95
EnemyLeft Favour	0.9
Action Mutation	0.09

Table 6: Mutation parameters of the LEMMEL learning run.

task instance, many of the game engine’s assets were held statically and were mutable. Moreover, the MarioEnvironment was implemented as a singleton. This meant only one task could be performed at a time, resulting in the evaluation stage being set to run in only one thread. Despite this setback, learning runs took approximately 6 hours to complete.

## 6.6 Parameters and Revisions

Many learning runs were performed during the project, including numerous parameter revisions. For brevity, only the most significant changes will be presented.

Initial runs used parameter files that closely followed the approach used by the REALM agent team. Conditions favoured the DONT\_CARE byte with a probability of 0.4, whereas actions were reset mutated. Every gene had a mutation probability of 0.2. Agents were evaluated over 10 levels in a similar style to REALM’s method. The levels varied over a wide range of difficulties, but favoured easier levels. Enemies were enabled for all but 2 levels. The evaluation multipliers rewarded distance, level completion, Mario’s final mode, kills and time left upon completion.

The agents produced with these parameters did not display any interesting behaviours and functioned similarly to the handcrafted Forward Jumping agent (4.5). the first revision aimed to address this.

Mutation probability of both conditions and actions was lowered and the DONT\_CARE condition favour probability was increased to 0.5. The biggest change came with the evaluation task. Agents now played a narrower band of difficulties, generally more difficult than the first set. Enemies were only enabled for the last 4 levels and pits were enabled for all. These changes were made as they suited a reactive agent over the blind forward jumping approach. Lastly, the level completion was given more weight.

Agents evolved with these parameter demonstrated much more compelling behaviours, as well as higher overall attainment. However, it was observed that much of their ruleset's were unused.

A second revision was made with an aim to increasing the number of rules used and to reduce the search space. Favour probability for genes corresponding to conditions on MarioMode and EnemyLeft were greatly increased, making DONT\_CARE likely in these conditions on all rules. For clarity, the learning run performed using these parameters will be referred to as the **LEMMEL** run (for **LE**ss MarioMode and **E**nemy**L**eft)

The agents produced with this approach are evaluated in depth on Sections 7.2 and 7.3. Tables 4, 5 and 6 summarise the LEMMEL parameters, which can be found in full, in ECJ's file format, in Appendix E.

## 7 Results

### 7.1 Learning Data

Statistical output from each generation of the LEMMEL run (the most significant and successful learning run, described in Section 6.6) was collated and organised. The information is presented in graphical form in Figure 11. It includes the best (red) and average (blue) fitness for each generation, with polynomial trend lines. The **complex** handcrafted agent (see Section 4.5) was also passed into the evaluation task of each generation. Its fitness is also included on the graph in green.

#### 7.1.1 Improvement over Time

The graph indicates that both best and average fitness of the learning agents improved over time. Average fitness increased gently over the first 150 generations, from below 10,000 to around 25,000, after which it levelled off, with some fluctuations. Best fitness increase rapidly over the first 150 generations, from around 15,000 up to just below 60,000. It then rose more gently until generation 500, reaching approximately the 62,000 mark. There was a slight dip in best fitness after generation 850, which is likely a result of high variance, which is explained in the the two proceeded sections.

The stagnation in improvement can be explained, in part, by the best individuals saturating the evaluation task. Scores above 60,000 are typical of very capable agents, who have completed 8 or 9 out of the 10 task levels. Scores over 70,000 represent a near perfect execution of the levels, completing them all in good time without many enemy collisions. We can see that, after generation 400, many populations contained individuals scoring in excess of 60,000, thus reducing the amount with which they could improve.

This is also reiterated by comparing the learnt agents' fitness with that of the complex agent. The trend line shows that the best performing agent of a generation often outperformed the handcrafted agent. Further analysis comparing the handcrafted agents with the those developed in the LEMMEL run is discussed in Section 7.2.

#### 7.1.2 Generation Variance

A feature very evident in the LEMMEL graph is the amount of fitness variance between successive generations, which can be seen in both the best and average fitness. Analysis of the polynomial regressions reveals the standard error of regression for these datasets is 5,612 and 3,287 respectively. These figures are high for an optimisation algorithm, especially given that truncation selection usually helps to reduce deviation [45, s. 3.8.3]. Moreover, as



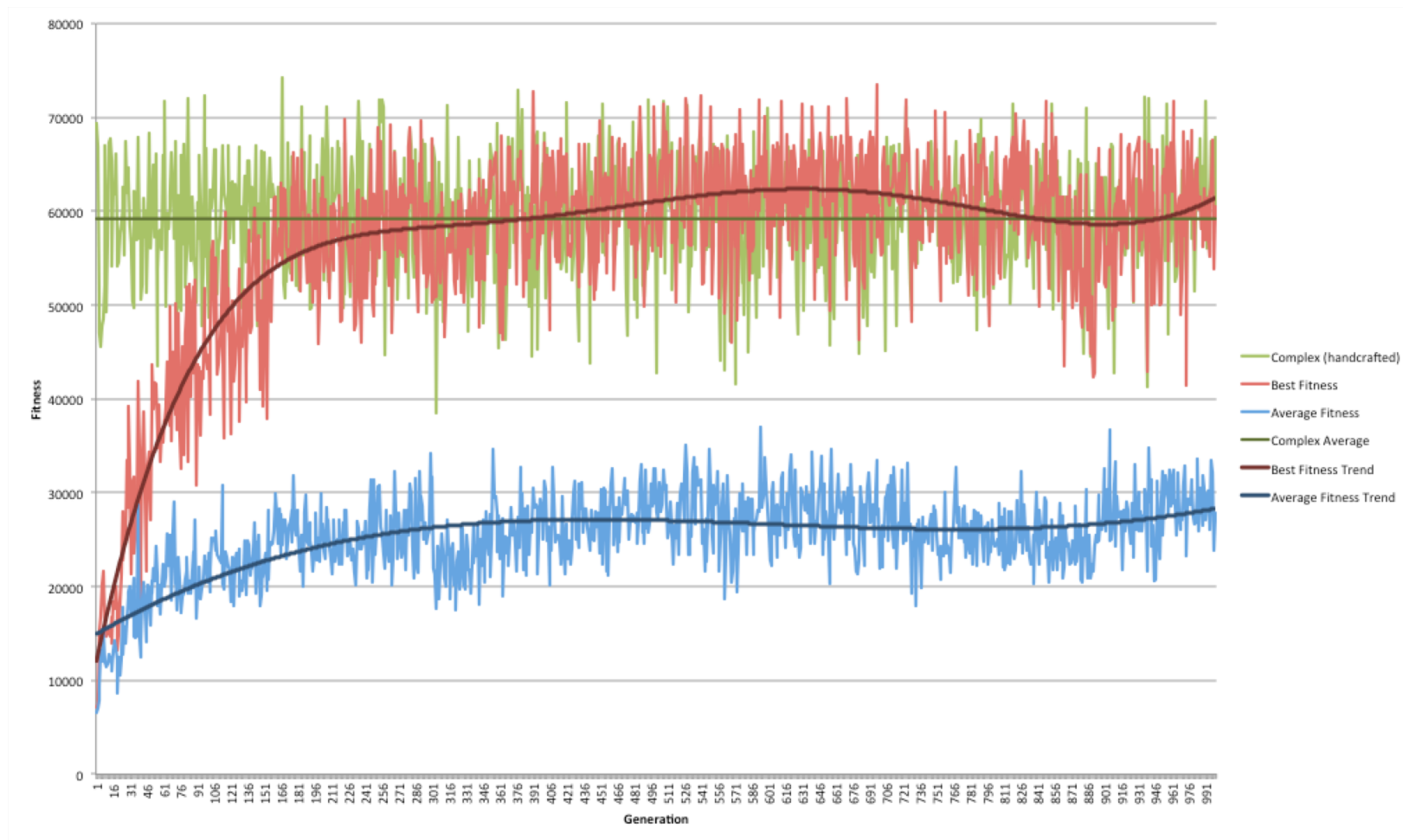


Figure 11: Graph charting the best and average fitness over the LEMMEL learning run. Handcrafted *complex* agent fitness also included.

Run	Average Fitness	Standard Deviation
LEMMEL	27,740	18,850
Fixed Seed	29,601	25,293

Table 7: Deviation over the fitness of the final population of the learning runs, demonstrating high population variance.

a  $(\mu + \lambda)$  evolutionary strategy is being used (i.e. the best performing individuals are carried over to the next generation), one would not expect the best fitness to decrease often. However, the data shows this occurs regularly.

This trait is due to the use of different level seed per generation, which means that each generation has a slightly different evaluation task. It was unexpected that the level seed would affect task difficulty (and therefore best fitness) as drastically as the data suggests.

To investigate this further, and to ascertain if it had any effect on the capability of the learn agent(s), a supplementary learning run was performed. This run had a single level seed, which was used in every generation, thus every generation had an identical evaluation task. For clarity, this run will be referred to as the **fixed-seed** run. The data from the fixed-seed run is charted in Figure 12.

As expected, the best fitness in this run only ever increased<sup>3</sup>. Moreover, the deviation in average fitness decreased, with a standard error of regression of 2,817. Furthermore, the highest fitness achieved was higher, which may mean either it is more capable than those evolved in the LEMMEL run or simply became very suited to the fixed set of levels. To determine this the agent will be included in the learnt agent analysis in Section 7.2, alongside those from the LEMMEL run.

### 7.1.3 Population Variance

Both the LEMMEL graph and the fixed seed graph show a large difference between best and average fitness. this suggests a high deviation over fitness within each population. This is confirmed by analysing the final generation of both runs, for which the full population statistics are available. The average fitness and standard deviation are shown in Table 7.

As each generation is constructed from the best individuals of the previous generation, such population variance shows that the ruleset genomes are highly sensitive to mutation. For example, we can consider the final population from the fixed seed run. The truncation selection of generation 999

---

<sup>3</sup>Except for after generation 882. This fitness evaluation is erroneous and an example of the occasional non-deterministic nature of the benchmark game-engine. Attempts to repeat the value (using that generations best agent) were unsuccessful and instead produced a value that was lower than the previous generations best fitness.

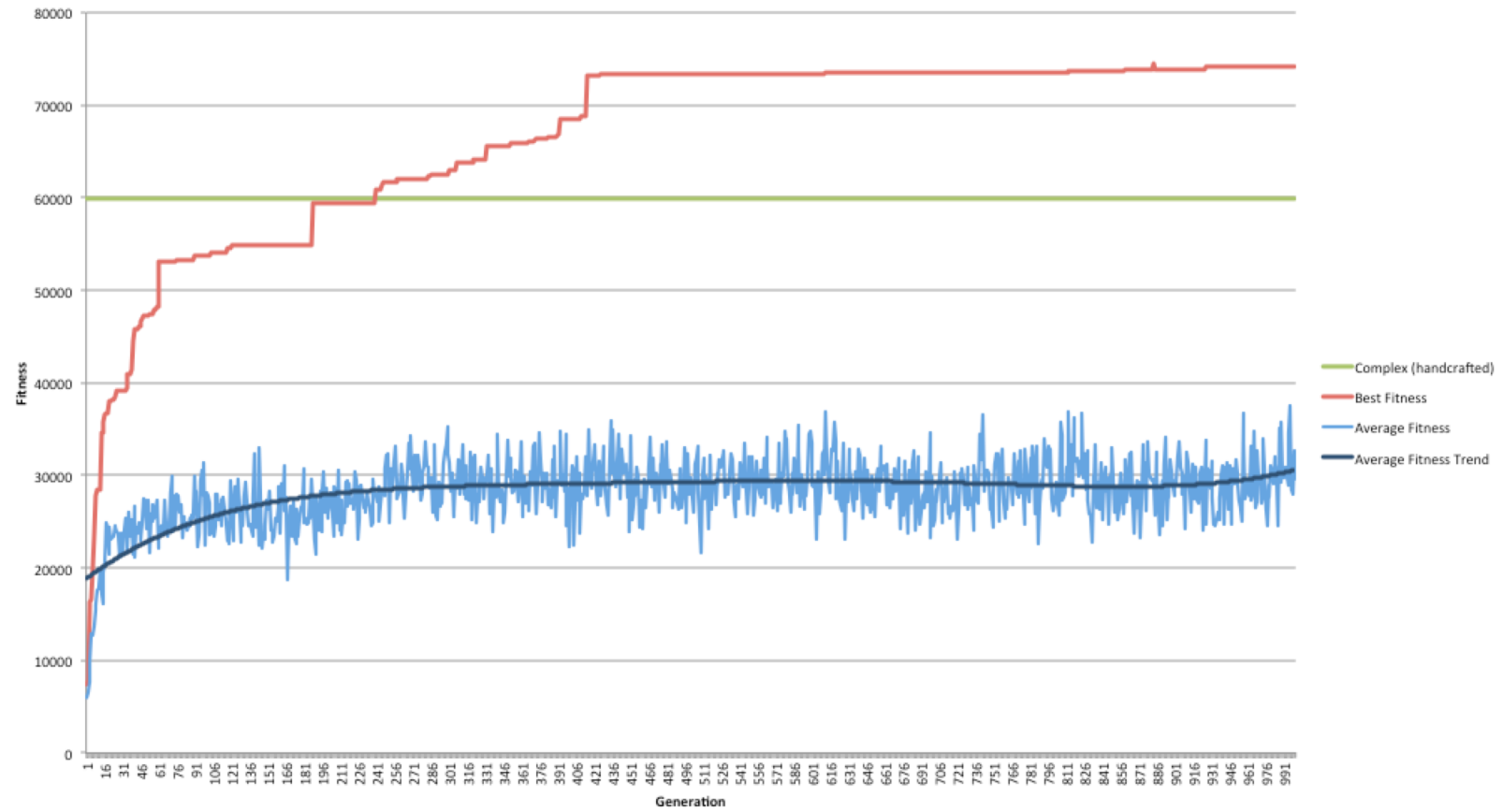


Figure 12: A learning run which had a fixed level seed, performed in response to high generational variation in the LEMMEL run.

restricted the parent pool to 5 individuals, each with a fitness of over 74,000. The generation 1000 population was made up entirely of their mutations, but still contained 23 (46%) individuals whose fitness was under 10,000.

More specifically, the best individual of generation 999 contained the following rule (a key for which can be found in Appendix A):

#	Conditions				Actions			
	ELR	OA	PB	MY	Left	Right	Jump	Speed
15	0	0	0	1	F	F	F	F

This rule states that when there are no enemies or obstacles in front and no pit below, Mario should do nothing. The was rarely used due to the condition that Mario must also be moving down: **MY** (MovingY) = 1. In one individual, a single gene mutation removed the condition on **MY**, this caused the rule to activate constantly at the beginning of each level, leaving Mario frozen. This caused the fitness to drop from 74,194 to 4,687, which shows that a single mutation can have a devastating affect on ruleset fitness.

## 7.2 Handcrafted vs. Learnt Agents

As explained in Section 6.3.5, the learning process was configured to extract three rulesets to be considered for evaluation. After generation 800, the algorithm saves the agent with the best overall fitness and the agent with the largest fitness compared to the average. The two such agents from the LEMMEL run (alongside the final agent from the fixed seed run) are compared to the handcrafted agents in this section. For reference, they will be named **learnt best**, **learnt difference** and **learnt fixed seed** agents respectively, and called the learnt agents collectively.

### 7.2.1 Learning Evaluation Task

The three learnt agents and three handcrafted agents were run through the each generation’s evaluation task of the LEMMEL run. The agents perform 1000 tasks, with the options and level seed from LEMMEL parameter file (full details can be found in Section 6.6. Results are compiled in to Table 8. We can see from the data that both the learnt best and learnt difference agents performed at the same level as the complex agent and considerably better than the other handcrafted agents. This implies they became very proficient at the task they learnt over. This is especially significant as the LEMMEL evaluation task was designed to favour the complex and simple reactive agents.

We can also see that the learnt fixed seed agent did not perform quite as well, suggesting it became too suited to the single set of levels it was evolved over. The large difference between each agent’s best and worst fitness reiterates that the use of different level seeds may be overly affecting

Agent	Average	Offset	Best	Worst
<b>Complex</b>	59,206	0	74,286	38,494
<b>Simple Reactive</b>	37,310	21,896	61,784	19,526
<b>Forward Jumping</b>	20,458	38,748	45,022	8,378
<b>Learnt Difference</b>	58,930	276	72,672	36,440
<b>Learnt Best</b>	59,079	127	73,058	43,740
<b>Learnt Fixed Seed</b>	54,115	5,091	74,194	38,494

Table 8: Statistics from handcrafted and learnt agents playing each generation’s evaluation task from the LEMMEL run.

Agent	Total Score	Levels Completed	Enemies Killed	Distance
<b>Complex</b>	1,758,931	168 (33%)	1,440 (8%)	62,231 (45%)
<b>Simple Reactive</b>	1,102,733	87 (17%)	588 (3%)	44,038 (32%)
<b>Forward Jumping</b>	954,141	65 (13%)	743 (4%)	38,484 (28%)
<b>Learnt Difference</b>	1,521,200	144 (28%)	1,130 (6%)	55,710 (40%)
<b>Learnt Best</b>	1,510,697	133 (26%)	1,603 (8%)	54,341 (39%)
<b>Learnt Fixed Seed</b>	1,267,914	102 (20%)	1,173 (6%)	48,131 (35%)

Table 9: Competitive statistics from handcrafted and learnt agents playing the comparator task with a seed of 1000.

difficulty. However, as the learnt fixed seed agent did not score as highly as the other learnt agents, it suggests that although the LEMMEL run had high generational variance, it produced more capable agents.

### 7.2.2 Comparator Task

The six agents were also passed into the more extensive comparator task (described in Section 5.5) with a seed of 1000. Each agents results are presented in Table 9. We can see from the data that both the learnt best and learnt difference agents performed better than the two simpler handcrafted agents, but not as well as the complex agent. Considering the evaluation task result, this shows that the learning evaluation task may not adequately represent the comparator task and the game more generally. The data also shows that the learnt fixed seed agent did not perform quite as well, suggesting that agents that learn from many sets of levels are more adept at tackling a greater variety of challenges.

Figure 13 charts how the agents performed at different difficulties in the comparator task. None of the agents performed well on difficulties over 10. Further analysis showed that they all failed to clear the large jumps which are common to these levels, demonstrating a weakness in the agent framework. Elsewhere, the complex agent outperformed the learnt agents on

#	Conditions											Actions			
	MM	JA	OG	EL	EUR	ELR	OA	PA	PB	MX	MY	Left	Right	Jump	Speed
1			1		1	1		0	1	2	0			T	T
2		0								2	2		T	T	T
3		1			1		1		0		0			T	T
4		0	0			0	1	0	1	0	1	T	T	T	T
5					0	0	1		0	0	0			T	
6		0			1	1	1		0	0				T	T
7		1	0			0		0	0		2				
8				0	0	0	0		1	2	1		T	T	T
9									0	0	2			T	
10		0	1				1	1		0					
11	0		1	1	0	1		1	0		1		T		T
12					1			2							T
13							0		0	2	1	T		T	T
14		1	0	1	0	1	1	0		0		T	T	T	T
15		1							1	0	2	T	T	T	T
16		1	1	0				1				T	T	T	
17		1	1		0				1		0		T	T	T
18		1	1			1		0						T	
19		1			1			2	0		0	T			T
20			0			0	0		0	0	0	T	T	T	T
-													T		T

Table 10: Ruleset for learnt difference agent. Blank entries denote *DONT\_CARE* for Conditions and **false** for Actions. A key for this table can be found in Appendix A

nearly all difficulties. However, both the learnt best and the learnt difference agents outperformed the complex agent on difficulty 2. This was the most prevalent difficulty in the LEMMEL evaluation task and suggests that a longer or more representative task may produce a more capable agent. We can also see that the disparity between the learnt fixed seed agent and its peers come mainly from the lower difficulty levels. At these difficulties, the other learnt agents take greater advantage of the reduced presence of enemies and pits. This is especially true of difficulty 0, where there are no pits and very simple enemies, in which the fixed seed agent was outperformed by all the other agents.

### 7.3 Learnt Agent Analysis

Of all the learnt agents, the learnt difference agent performs the best on the comparator task. It also demonstrates the most interesting behaviour, which is analysed in this section. A full ruleset for this agent can be found in Table 10, a key for the table can be found in Appendix A.

#### 7.3.1 Rule usage

The agent only regularly uses 7 of its 20 rules (35%), and relies heavily on the default action. On one hand, some of its rules contain conditions that are too specific, for example, Rule 11, which is never used. On the other,

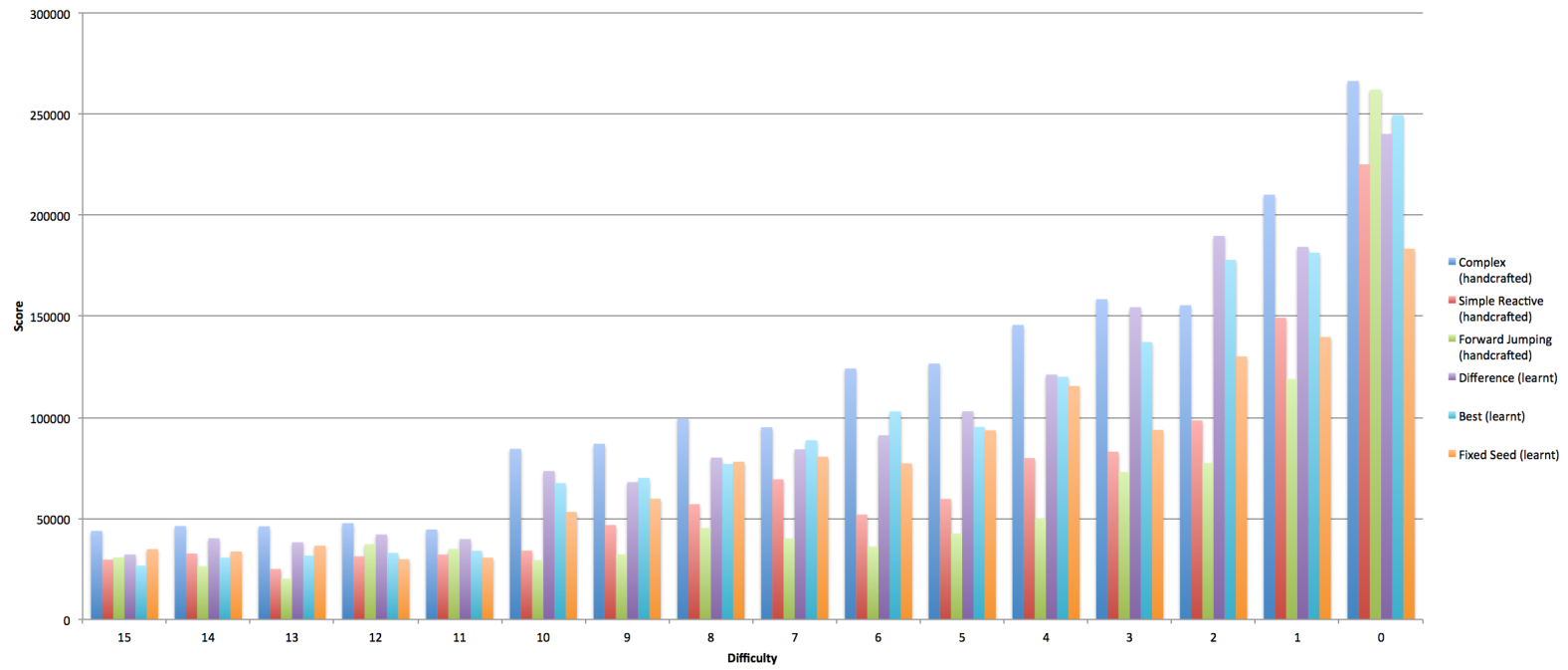


Figure 13: Graph charting the score achieved on different difficulty levels during the comparator task.

some rules are not specific enough and are eclipsed by others, for example, Rule 12, which is often forgone in favour of Rule 19. Rather than mutation or favour rate (see Section 6.1) being too high or too low, this suggests that the design of the genome mutation itself is too simplistic. Methods of addressing this are discussed in the Section 8.3.

### 7.3.2 Behaviours

The learnt difference agent acts similarly to the complex and simple reactive agents; it attempts to jump over obstacles, pits and enemies when it detects them and runs at speed otherwise. It struggles with the largest pits and avoiding collisions when enemy concentration is high. Moreover, much like the complex agent, it occasionally gets stuck in an action loop and runs out of time.

Four significant and interesting behaviours, and the rules that enable them, are presented subsequently.

#### Enemies

#	Conditions								Actions			
	JA	OG	ELR	OA	PA	PB	MX	MY	Left	Right	Jump	Speed
2	0						2	2	F	T	T	T
13				0		0	2	1	T	F	T	T
18	1	1	1		0				F	F	T	F
-									F	T	F	T

To avoid colliding with enemies the agent uses Rules 2, 13 and 18, together with the the default action. These rules are pictured above with the final entry denoting the default action/rule. A key for these rules can be found in Appendix A.

Figure 14 illustrates this behaviour. Mario approaches the enemy using the default action. On detecting the enemy (**ELR** (EnemyLowerRight) = 1) Rule 18 activates and Mario jumps. Mario is now travelling up and right (**MX** (MovingX) = 2, **MY** (MovingY) = 2), thus Rule 2 activates to continue the jump and move him quickly to the right. Rule 2 also has the side effect of shooting a fireball if Mario is in **fire** mode. Rule 13 activates as Mario starts to descend (**MY** = 1), turning Mario left and slowing the jump. This allows the fireball to beat Mario to the landing zone, killing any enemies that might pose an immediate threat.

Enemy avoidance is the weakness of several of the learnt agents. The difference agent's approach of jumping high over them and shooting a fireball is one of the most effective. Using Rule 13 to slow the jump once the enemy is cleared allows Mario to avoid jumping into subsequent enemies or pits.





(a) Default Rule



(b) Rule 18



(c) Rule 2



(d) Rule 13



(e) Default Rule



(f) Default Rule

Figure 14: Learnt Difference agent tackling enemies.

## Obstacles

#	Conditions						Actions			
	EUR	ELR	OA	PB	MX	MY	Left	Right	Jump	Speed
5	0	0	1	0	0	0	F	F	T	F
9				0	0	2	F	F	T	F
13			0	0	2	1	T	F	T	T
-							F	T	F	T

In order to overcome obstacles, such as level terrain and blocks, the agent uses Rules 5 and 9, in conjunction with the default action.

The process is demonstrated in Figure 15. The agent approaches the wall using the default action. On hitting the obstruction, Mario stops moving (**MX** (MovingX) = 0, **MY** (MovingY) = 0) and detects the obstacle (**OA** (ObstacleAhead) = 1). This activates Rule 5, causing Mario to jump. As Mario is now moving straight upwards (**MX** = 0, **MY** = 2) Rule 9 is activate and ensures Mario continues the jump to its maximum height. When descending, Mario alternates between using Rule 13 and the default action. This results in Mario moving slightly to the right and landing on the obstacle, successfully clearing it.

The behaviour is particularly interesting due to the condition on Mario being stationary before jumping. This allows Mario to clear obstacles without over-jumping. Over-jumping could cause a him to fall into a proceeding pit, especially when climbing ‘stairs’ (see Figure 16).

This is an example of a behaviour not considered during the construction of the handcrafted agents. The complex agent handles the issue of over-jumping by detecting potential problems and moving left.

Many evolved rulesets, through several of the learning runs, demonstrate this behaviour, confirming its benefit to the agent. Furthermore, during the LEMMEL run, it is one of the earliest to develop, appearing first at around generation 50.



(a) Default Rule



(b) Rule 5



(c) Rule 9



(d) Rule 9



(e) Default Rule



(f) Default Rule

Figure 15: Learnt Difference agent tackling obstacles, without over-jumping.

## Pits

#	Conditions										Actions			
	JA	OG	EL	EUR	ELR	OA	PA	PB	MX	MY	Left	Right	Jump	Speed
2	0								2	2	F	T	T	T
8			0	0	0	0		1	2	1	F	T	T	T
13						0		0	2	1	T	F	T	T
16	1	1	0				1				T	T	T	F
-											F	T	F	T

To jump over pits Mario uses Rules 2, 8, 13 and 16, along with the default action.

Mario can be seen jumping over a pit with ‘stairs’ in Figure 16. If the pit does not have stairs then Mario approaches using the default action, otherwise, he climbs them one by one using his obstacle behaviour. As soon as he lands on the top step (**JA** (JumpAvailable) = 1, **OG** (OnGround) = 1) he detects that the pit is close (**PA** (PitAhead) = 1). This activates Rule 16, causing Mario to jump. Rule 2 then activates to maximise the jump (as we saw with enemies). When Mario reaches the top of the jump and begins to descend (**MY** (MovingY) = 1) Rule 2 deactivates. If Mario is still above the pit (**PB** (PitBelow) = 1) Rule 8 is used to ensure Mario reaches as far as possible. As soon as Mario is no longer over the pit (**PB** = 0) Rule 13 is used to slow the jump and avoid over-jumping. If Rule 13 is used to the point that Mario stops moving right (**MX** (MovingX) = 0 or 1) the default action is used instead, which helps Mario avoid turning back into the pit. Often, this means that Mario lands the very edge of the pit, at which point he uses the default action to continue.

Pits are the most dangerous part of any level, as they can cause an instant loss. Hence, clearing them is vitally important for the capability of an agent. The learnt agent’s strategy of always maximising the jump whilst it is above a pit means that it clears the majority of them. Furthermore, its ability to land on the edge of the right-hand side of the pit allows Mario to successfully avoid over-jumping and tackle the next hurdle.

Every learnt agent displays some level of this behaviour. However, most of them are negatively affected by the proximity of enemies, whereas the learnt agent’s pit strategy is less concerned with their presence. This is important as a pit can cause instant failure, whereas an enemy collision is only fatal when in **small** mode. In fact, the learnt agent also displays an additional positive behaviour in regards to the combination of enemies and pits, which is explained next.



(a) Default Rule



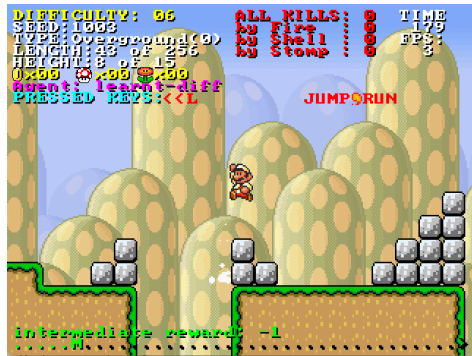
(b) Rule 16



(c) Rule 2



(d) Rule 8



(e) Rule 13



(f) Default Rule

Figure 16: Learnt Difference agent tackling pits, without over-jumping.

## Pits with Enemies

#	Conditions							Actions			
	JA	OG	EL	EUR	PA	PB	MY	Left	Right	Jump	Speed
16	1	1	0		1			T	T	T	F
19	1			1	2	0	0	T	F	F	T
-								F	T	F	T

In situations when the path over a pit is blocked by an enemy, the learnt difference agent displays an advanced strategy. It delays jumping over the pit until the enemy had moved away using Rule 19.

Figure 17 demonstrates the strategy. As Mario approaches a pit he first detects that one is **far** (**PA** (PitAhead) = 2), as he gets closer he detects it as **close** (**PA** = 1). If there is an enemy to his upper right (**EUR** (EnemyUpperRight) = 1) at the point he perceives the pit as **far** Rule 19 activates, which stops Mario and causes him to move left. The use of the **speed** action here, allows Mario to turn quick enough to avoid perceiving the pit as **close** and activating Rule 16. Turning and moving away from the pit allows the threat to either move away or fall into the pit (as can be seen in Figure 17). At which point Rule 19 deactivates and Mario once again approaches the pit. If no more threats are detected, he uses the pit strategy to clear the pit. The other conditions on Rule 19 also serve an important purpose: it will not occur if Mario has already started to jump over the pit. Turning left whilst already trying to clear a pit would more than likely result in Mario falling and failing the level.

The use of Rule 19 allows Mario to avoid collisions with enemies, maintaining the **fire** mode, allowing him to use fireballs. As we saw in his enemy strategy, fireballs are an important tool for clearing enemies, increasing the chances of completing a level.

This behaviour is unique to the learnt difference agent and is part of what makes it the most interesting and capable agent.



(a) Default Rule



(b) Default Rule



(c) Rule 19



(d) Rule 19



(e) Default Rule



(f) Rule 16

Figure 17: Learnt Difference agent tackling pits where enemies are blocking the jump.

## 8 Project Evaluation

With the results presented above, we can now critically consider the project in terms of the requirements set out in the project’s aim, objectives (Section 1.3) and specification (Section 3).

### 8.1 Agent Framework

The primary objective of the agent framework was to enable and encourage meaningful improvement during learning. The focus on a rule based system allowed for agents to be represented with a simple encoding, which afforded a great deal of freedom to the learning module. It reduced development time as several features of the ECJ library could be used as is and any extensions did not have to process a complex data structure. Moreover, it allowed for greater control, which was crucial in honing and revising the learning parameters. Ultimately, we can see from the results that, given the right parameters, the agent framework did enable significant improvement over generation, with later agents achieving near maximum fitness.

However, the results show a high fitness deviation over each generation’s population. This is likely due to the inherent sensitivity to mutation of a rule-based approach. One gene mutation can transform a once useful rule into one that is counter productive, with a great affect to the agent’s capability.

In addition to the project’s aims and objective, three criteria were laid out for the agent’s use of a rule based system in Section 3: speed, capability and complexity. The framework design relies heavily on how the agent senses the environment; as such the selection of perceptions is a major factor in its evaluation.

The **complex** handcrafted ruleset shows that the framework’s choice of perceptions allows for capable agents. It rarely fails to finish the easier levels and can successfully tackle the more difficult ones. Furthermore, the depth of the perceptions and the available actions allows for competent behaviour to be demonstrated by relatively small rulesets, which improves the agent’s speed. In fact, agent response time never rises above a millisecond. This allows for levels to be run at practically any number of frames per second and has little effect on total learning time.

On the other hand, the framework is still limited. It is impossible for any ruleset agent to break itself out of a loop or to navigate a dead end, as they have no memory. Similarly, it can never develop a truly effective enemy killing behaviour without differentiating between types.

However, as discussed in Section 4.1, improving capability by embellishing the agent’s perceptions comes at the penalty of complexity. A more complex design lends to a larger search space for the learning algorithm, which hinders the development of a successful agent through random mu-



tation. This was also a crucial consideration of the REALM team, whose V2 agent was designed, in part, to reduce the search space of the VI agent [35][p. 86].

In its current form, the agent framework balances these factors reasonably well. Our results show that a successful and interesting agent can be evolved despite the total number of possible rules

$$4^4 * 3^7 * 2^4 = 8,957,952$$

being higher than that of REALM’s V1 agent (7,558,272) [35][p. 86]. Thus, improvements to the capability of the agent should not increase the search space, but instead focus on making sure each perception is purposeful.

In the LEMMEL run parameters (and in the handcrafted agents) use of the MarioMode and EnemyLeft perceptions was discouraged. Our learning parameters allowed us to reduce the search space by increasing the probability of the DONT\_CARE condition. This is clearly a counterintuitive approach and the agent design should be amended by removing these perceptions.

Such removals would allow for additional perceptions and/or design extensions without increasing agent complexity. Given more time, a redesign that allowed agent encoding control over its perceptions would be considered. Perceptions would remain fixed, but the parameters that control them would be variable. For example, the area in which an agent looks for an obstacle or enemy could be controlled by parameters on a ruleset wide basis. Moreover, perceptions that measure the last used action (a simplistic form of memory) and the identification of enemies in greater detail would also be investigated.

## 8.2 Level Playing

The primary motivation for the level playing module was to meet Objective 4: Provide a large and diverse test bed from which to learn. This was reiterated in the project specification (Section 3), which decrees that it must, in a parametrised manner, produce a variety of levels and ways to score an agent.

The former was achieved by providing access to the benchmark’s level generation options through simple data classes. Additionally, as detailed in Section 5.3, modifications were made to the LevelGeneration class to provide greater diversity and to ensure parameters were being treated correctly. This was an unexpected workload, and as such could not be afforded ample attention. Without time constraints, further modifications would have been made, including improved scaling of enemy types and better block placement. Furthermore, these and previous changes, as well as other level generation features, would have been made parametrisable, allowing the MWLevelOptions class more power and precision.

The latter was achieved by the `MWEvaluationMultipliers` class, which parametrised a linear function over level playing statistics. The options mirrored the statistics produced by the benchmark software. Whilst this was effective in calculating agent fitness in the learning module, additional statistics would be beneficial. For example, it is currently impossible to punish an agent for the manner in which it failed (i.e. by falling into a pit, by colliding with an enemy, or by running out of time). If the project was to continue, the reliance of the benchmark software for these statistics would be reduced and a specialised `EvaluationInfo` class would be implemented, allowing for the level playing statistics to be extended.

Specifying and persisting these two elements was accomplished by extending ECJ's parameter file system. Adopting this existing system saved time and ensured it integrated well with the learning module. With the high number of revisions and the emergent importance of the evaluation task during the project, this decision proved vital. Whilst it provided great control when using a small number of levels, it was too verbose to be used for larger level tasks. For example, the comparator task (Section 5.5 and Appendix D) had to be hard-coded rather than specified in a level options file. With more time, the vocabulary of the parameter persistence would be extended to include controls for segments of episodes and values as a function of episode number (as seen in the comparator task).

The results demonstrated that the learning runs featured a high level of inter-generational variance. This is to be expected, especially of the average fitness, due partly to the agent framework's sensitivity to mutation. However, the variance is also displayed by the handcrafted agent, which suggests that it may in fact be caused by the difficulty of the evaluation task fluctuating between generations. As the only task parameter that changes between generations, the choice of level seed may be having too much of an effect on level generation. The next subsection will discuss ways in which this issue can be tackled within the parameters of the learning algorithm. However, if given additional time, the influence of the level seed on level difficulty would be investigated.

Aside from level generation, the benchmark software also raised several other barriers to the project. Firstly, there was little to no documentation, which slowed the initial progress of the project. Secondly, it greatly hindered the testability of the project as a whole (as discussed in Section 5.4), making it very difficult to provide significant test coverage (as prescribed by Objective 3). Lastly, it impeded the project's ability to meet the non-functional requirement on thread safety. As mentioned in Section 6.5, the game-engine is implemented as a singleton, relying heavily on static assets. A continued project would look to refactor this and allow the evaluation stage of the learning process to be multithreaded.

### 8.3 Learning Process

Section 6.6 described the revision of the learning parameters during the project. The results show that honing these parameters was crucial to the success of the project. Due to time constraints, parameter adjustment had to be stopped to allow for results to be gathered and analysed. Without such a restriction more parameter revisions would have been made.

Firstly, a continued attempt to reduce generational variance (in conjunction with solutions mentioned in the previous two sections) would be made. We saw that fixing a single level seed for the entire run removed variance from the best fitness (as expected) and lowered variance in the average fitness. However, it produced a less capable agent, which was overly suited to the small number of levels it played (known as overfitting). Instead a compromise would be struck, perhaps taking seeds from a fixed pool, or have them run in generational segments. In this way agents are evaluated over a wider range of levels, but not at a detriment to the learning evolutionary process.

Secondly, reducing overfitting in general would be investigated. We saw in the results that the learnt agent performed better than the complex hand-crafted agent at the learning evaluation task, but did not perform as well on the comparator task. We also saw that in both the fixed and variable seed run, the learnt agent achieved near perfect score relatively early, suggesting the task could be made more difficult. In future revisions, the learning task would be run on higher difficulties, with more focus on enemies. Also, with the ability to multithread the evaluation, a more extensive task could be run, without a time penalty. Increasing the number of levels would allow the task to be more representative of the comparator task.

Even without the above revisions, we have seen that the use of an Evolutionary Strategy has proved successful. As a result, both the aims and the specification of the project were widely met. Meaningful learning was observed and the final agent demonstrated interesting behaviours, including some that were unforeseen and not considered during the creation of hand-crafted rulesets. Moreover, as the use of truncation selection is known to reduce variance in learning algorithms [45, s. 3.8.3], and due to the variance induced by both the agent framework’s mutation sensitivity and the level playing’s seed variation, the choice of an evolution strategy turned out to be very appropriate.

However, because of the scope of the project, its implementation is quite simplistic. With more time, several additional features would be considered. Firstly, the use of rule crossover would be investigate, as seen in the REALM team’s approach (described in Section 2.2.3). Secondly, utilising a Tabu list, a common approach in optimisation search algorithms [46], would be studied. Rules that caused a significant drop in fitness would be added to a blacklist and would be avoided by future mutations. Lastly, adaptive

mutation rates, a common feature of ESes [19, s. 4], would be examined. The results showed that the learnt agent only used a small percentage of their ruleset. In order to combat this, mutation rates could change during a learning run on a rule by rule basis. If a rule goes unused in the evaluation stage, then its mutation rate is increased. However, if it is used often, then its mutation rate is decreased.

The results that were gathered from the statistical output of the learning module have proved a good analysis of the learnt agent and the process in general. However, it was impossible to properly ascertain key metrics such as population variance and mutation sensitivity. With more time, the fitness and genome of each individual in the learning run would be made available, allowing for a more in depth analysis.

## 8.4 Methodology

Overall, the project achieved the majority of its objectives and realised most of the project specification. Its failure to attain all of its aims was centred around two aspects: testing and testability and issues with the Mario benchmark software.

Testing proved more difficult and time consuming than anticipated, due in part to the inclusion of ECJ and the benchmark (see Sections 4.4, 5.4 and 6.4). Testing was infeasible for several parts of the level playing and learning modules. This precipitated the abandoning of test driven development (Objective 2) and reduced the test coverage (Objective 3) of the entire project. If the project were to be attempted again, more time and effort would be spent making the code more testable.

The Mario benchmark software included several problems (see Section 5.3). Those that were fixed, were time-consuming and caused a schedule overrun. Whereas those that could not be addressed, lead to the project specification not being fully adhered to (e.g. multithreading the learning process as discussed in Section 6.5). More research should have gone into the source code at an earlier date, which would have helped identify these issues sooner and allow for preparations to be made.

Due to the modifications of the benchmark software, comparisons to other approaches were limited. Framing the learnt agent in terms of the Mario AI Competition would have been misleading and uninformative as it used an engine that produced much more difficult levels. However, greater effort should have gone into making these comparisons, especially to the REALM agents. Given more time, the learnt agent would have performed tests in the original engine, allowing score comparisons to the REALM agent and other competition entrants.

Despite these issues, the project met its overarching objective. It successfully created an interesting and capable game-playing agent, developed by evolutionary computation.

# Appendices

## A Perceptions

	0	1	2
<b>MarioMode</b> (MM)	small	big	fire
<b>JumpAvailable</b> (JA)	false	true	-
<b>OnGround</b> (OG)	false	true	-
<b>EnemyLeft</b> (EL)	false	true	-
<b>EnemyUpperRight</b> (EUR)	false	true	-
<b>EnemyLowerRight</b> (ELR)	false	true	-
<b>ObstacleAhead</b> (OA)	false	true	-
<b>PitAhead</b> (PA)	none	close	far
<b>PitBelow</b> (PB)	false	true	-
<b>MovingX</b> (MX)	none	left	right
<b>MovingY</b> (MY)	none	down	up

Table A1: Displays the different perceptions and what their byte value represents.

```
// Returns NONE if there are no pits ahead, FAR and CLOSE if there is one in
// a certain number of columns relative to mario.
case object PitAhead extends BytePerception(7, 2) {
  val NONE: Byte = 0; val CLOSE: Byte = 1; val FAR: Byte = 2;
  val COL_CLOSE_L = 1; val COL_CLOSE_R = 1; val COL_FAR_L = 2; val
    COL_FAR_R = 2
  def apply(environment: Environment): Byte = {
    val pitOp: Option[Int] =
      Perception.getOpens(environment, COL_CLOSE_L, COL_FAR_R).headOption
    pitOp match {
      case Some(x) if (x >= COL_CLOSE_L && x <= COL_CLOSE_R) => CLOSE
      case Some(x) if (x >= COL_FAR_L && x <= COL_FAR_R) => FAR
      case _ => NONE
    }
  }
}

def getOpens(environment: Environment, a: Int, b: Int): List[Int] = {
  val level = environment.getLevelSceneObservationZ(2);
  val test = (x: Byte) => x == 0 || x == GeneralizerLevelScene.COIN_ANIM;
  val bottomRow = level.length - 1
  val mario = getMarioPos(environment)
  val left = max(0, min(a, b) + mario._2)
  val right = min(level(0).length, max(a, b) + mario._2)
  var opens: List[Int] = left to right toList

  for {
    i <- mario._1 + 1 to bottomRow
    if (!opens.isEmpty)
  }{
    opens = opens.filter { j => level(i)(j) == 0 || level(i)(j) ==
      GeneralizerLevelScene.COIN_ANIM }
  }
  opens.map { x => x - mario._2 }
}
```

Listing A1: Pit detection in the perceptions classes.

## B Handcrafted Agent Rulesets

Included below are the full rulesets for the project’s handcrafted agents.  
Blank entries denote a DONT\_CARE condition or a FALSE action.

#	Conditions											Actions			
	MM	JA	OG	EL	EUR	ELR	OA	PA	PB	MX	MY	Left	Right	Jump	Speed
1		0	1										T		
-													T	T	T

Table B1: Ruleset for handcrafted Forward Jumping Agent.

#	Conditions											Actions			
	MM	JA	OG	EL	EUR	ELR	OA	PA	PB	MX	MY	Left	Right	Jump	Speed
1		0	1										T		
2						1							T	T	
3							1						T	T	
4								1					T	T	T
5									1				T	T	T
-													T		

Table B2: Ruleset for handcrafted Simple Reactive Agent.

#	Conditions											Actions			
	MM	JA	OG	EL	EUR	ELR	OA	PA	PB	MX	MY	Left	Right	Jump	Speed
1									1				T	T	T
2		1	1				1	2	0				T	T	
3		0	0				1	2	0				T		
4		1	1				0	2	0				T		T
5		1	1				0	1	0	2			T	T	T
6		0	1					1	0	2			T		T
7		0	0					2	0	2		T			
8		0	0					1	0	2	1	T			T
9		0	0					1	0				T	T	T
10			0			1		0	0	2	1	T			T
11		0	1			1		0	0				T		
12						1		0	0				T	T	
13			0			1		0	0				T	T	T
14					1		1	0	0			T		T	
15		0	1				1	0	0				T		
16							1	0	0				T	T	T
17							1	0	0	2			T	T	
18			1										T		T
-													T		

Table B3: Ruleset for handcrafted Complex Agent.

## C Full Level and Evaluation Options

```
/**
 * Options that control level generation
 */
class MWLevelOptions(
    val blocks: Boolean, // Blocks appear
    val cannons: Boolean, // Cannons appear
    val coins: Boolean, // Coins appear
    val deadEnds: Boolean, // Dead ends appear in terrain forcing Mario to
        turn back
    val enemies: Boolean, // Enemies/Creatures appear
    val flatLevel: Boolean, // Level is flat, no change in elevation
    val frozenCreatures: Boolean, // All creatures don't move
    val pits: Boolean, // Pits appear
    val hiddenBlocks: Boolean, // Hidden blocks appear
    val tubes: Boolean, // Tubes/Pipes appear
    val ladders: Boolean, // Ladders appear
    val levelDifficulty: Int, // Difficulty of level, effective range 0-25,
        0 easiest
    val levelLength: Int, // Length of level in blocks
    val levelType: Int, // Type of level, 0-Outside 1-Cave, 2-Castle
    val startingMarioMode: Int, // Mode Mario starts as 0-small, 1-big, 2-
        fire
    val timeLimit: Int // Number of Mario seconds allowed to complete level
)

/**
 * Multipliers for several level playing statistics.
 * Comments describe the statistic. For example, if Mario
 * completes the level It will be win * 1, otherwise it will win * 0
 */
class MWEvaluationMultipliers(
    val distance: Int, // Distance travelled by Mario in pixels (16 pixels
        to a block)
    val win: Int, // 1 for level complete, 0 otherwise
    val mode: Int, // Mario's final mode on completion or death, 2-fire, 1-
        big, 0-small
    val coins: Int, // Number of coins collected
    val flowerFire: Int, // Number fire flowers collected
    val kills: Int, // Number of enemy kills
    val killedByFire: Int, // Number of kills by fireball
    val killedByShell: Int, // Number of kills by shell
    val killedByStomp: Int, // Number of kills by stomp
    val mushroom: Int, // Number of mushrooms collected
    val timeLeft: Int, // Mario seconds left on completion, 0 if level not
        completed
    val hiddenBlock: Int, // Number of hidden blocks hit
    val greenMushroom: Int, // Number of green mushrooms collected
    val stomp: Int // Unused
)
```

*Listing C1: Full field definitions for MWLevelOptions and MWEvaluation-Multipliers described in Section 5.2.1*

## D Comparator Task Options

```
val defaultEvaluationMultipliers = new MWEvaluationMultipliers(  
    1, //Distance  
    2048, //Win  
    16, //Mode  
    16, //Coins  
    64, //FlowerFire  
    58, //Mushroom  
    42, //Kills  
    4, //KilledByFire  
    17, //KilledByShell  
    12, //KilledByStomp  
    8, //TimeLeft  
    24, //HiddenBlock  
    58, //GreenMushroom  
    10) //Stomp  
  
val compBaseOptions: MWLevelOptions = new MWLevelOptions(  
    true, //blocks  
    true, //cannons  
    true, //coins  
    false, //deadEnds  
    true, //enemies  
    false, //flatLevel  
    false, //frozenCreatures  
    true, //gaps  
    false, //hiddenBlocks  
    false, //tubes  
    false, //ladders  
    0, //levelDifficulty  
    256, //levelLength  
    0, //levelType  
    2, //startingMarioMode  
    200) //timeLimit  
  
def compUpdate(levelSeed: Int): (Int, MWLevelOptions) => MWLevelOptions = (i  
    : Int, options: MWLevelOptions) => {  
    options.withLevelLength(  
        (((i+levelSeed) * 431) % (501+levelSeed) ) % 462) + 50)  
        .withTimeLimit((options.levelLength * 0.7).toInt)  
        .withLevelType(i % 3)  
        .withLevelDifficulty((compNumberOfLevels - i)/32)  
        .withPits(i % 4 != 2)  
        .withCannons(i % 6 == 2)  
        .withTubes(i % 5 == 1)  
        .withCoins(i % 5 != 0)  
        .withBlocks(i % 6 != 2)  
        .withLadders(i % 10 == 2)  
        .withFrozenCreatures(i % 3 == 1)  
        .withEnemies(!(i % 4 == 1))  
        .withStartingMarioMode(  
            if (i % 7 == 5 || i % 7 == 1) {  
                if (i % 2 == 0) 0 else 1  
            } else 2)  
    }  
}
```

*Listing D1: Parameter classes for the comparator task described in Section 5.5*



## E LEMMEL Learning Parameter File

```
parent.0 = @ec.es.ESDefaults es.params

#General
breedthreads = 4
evalthreads = 1
seed.0 = 1
seed.1 = 909
seed.2 = 499311
seed.3 = 90032

# +++ ES +++
breed = ec.es.MuPlusLambdaBreeder
es.mu.0 = 5
es.lambda.0 = 45
generations = 1000

# +++ POP +++
pop.subpops = 1
pop.subpop.0.size = 50
pop.subpop.0.species = com.montywest.marioai.learning.ec.vector.
    RulesetSpecies
pop.subpop.0.species.fitness = ec.simple.SimpleFitness

# Rulelength 15, So 20 rules gives 300
pop.subpop.0.species.genome-size = 300
pop.subpop.0.species.ind = ec.vector.ByteVectorIndividual

# These will be ignored, but warnings otherwise
pop.subpop.0.species.min-gene = -1
pop.subpop.0.species.max-gene = 2
pop.subpop.0.species.mutation-type = reset
pop.subpop.0.species.mutation-prob = 0.0
pop.subpop.0.species.crossover-type = one

# Handles min and max
pop.subpop.0.species.dynamic-param-class = com.montywest.marioai.
    learning.ec.params.RulesetParams

# Condition params
pop.subpop.0.species.condition = true
pop.subpop.0.species.condition.mutation-prob = 0.05
pop.subpop.0.species.condition.favour_byte = -1
pop.subpop.0.species.condition.favour_probability = 0.5

# MarioMode Condition
pop.subpop.0.species.condition.0.favour_probability = 0.95

# EnemyLeft Condition
pop.subpop.0.species.condition.3.favour_probability = 0.9
```

```

# Action params
pop.subpop.0.species.action          = true
pop.subpop.0.species.action.mutation-prob  = 0.09

# +++ STATS +++
stat.num-children                    = 1
stat.child.0                        = com.montywest.marioai.learning.ec.stats.
    RulesetEvolveStatistics
stat.child.0.gen-file                = ../lemmel-gen.stat
stat.child.0.final-file              = ../lemmel-final.stat
stat.child.0.final-agent-file        = lemmel-final
stat.child.0.best-agent-file         = lemmel-best
stat.child.0.diff-agent-file         = lemmel-diff
stat.child.0.best-agent-limit        = 800
stat.child.0.diff-agent-limit        = 800

# +++ MUTATION +++
pop.subpop.0.species.pipe.source.0   = ec.es.ESSelection
pop.subpop.0.species.pipe            = com.montywest.marioai.learning.ec.
    vector.breed.RulesetMutationPipeline

# +++ EVAL +++
eval.problem                        = com.montywest.marioai.learning.ec.eval.
    AgentRulesetEvaluator

# Seeds for generating levels
# each generation g, seed used is:
# prev_seed + add + g*mult
# where prev_seed is seed_start on g = 0
eval.problem.seed                   = true
eval.problem.seed.start              = 3348
eval.problem.seed.add                = 284839
eval.problem.seed.mult               = 2568849

# Evaluation Multiplier
eval.problem.mults                   = true
eval.problem.mults.distance          = 1
eval.problem.mults.win               = 3200
eval.problem.mults.mode              = 200
eval.problem.mults.kills              = 100
eval.problem.mults.time-left         = 2

# Fallback Action
eval.problem.fallback-action         = true
eval.problem.fallback-action.right   = true
eval.problem.fallback-action.speed   = true

```

```

# Levels
eval.problem.level      = true
eval.problem.level.num-levels = 10

eval.problem.level.base.dead-ends = false
eval.problem.level.base.enemies   = false
eval.problem.level.base.cannons   = false
eval.problem.level.base.pipes     = false
eval.problem.level.base.start-mode-num = 2
eval.problem.level.base.length-num = 200
eval.problem.level.base.time-limit = 100
eval.problem.level.0.difficulty-num = 2
eval.problem.level.0.type-num      = 1
eval.problem.level.1.difficulty-num = 3
eval.problem.level.1.type-num      = 0
eval.problem.level.2.difficulty-num = 5
eval.problem.level.2.type-num      = 2
eval.problem.level.3.difficulty-num = 10
eval.problem.level.3.type-num      = 0
eval.problem.level.4.difficulty-num = 2
eval.problem.level.4.flat          = true
eval.problem.level.5.difficulty-num = 7
eval.problem.level.6.difficulty-num = 2
eval.problem.level.6.flat          = false
eval.problem.level.6.enemies       = true
eval.problem.level.6.frozen-enemies = true
eval.problem.level.7.difficulty-num = 2
eval.problem.level.7.frozen-enemies = false
eval.problem.level.8.difficulty-num = 3
eval.problem.level.8.type-num      = 1
eval.problem.level.8.tubes         = true
eval.problem.level.9.difficulty-num = 5
eval.problem.level.9.type-num      = 0
eval.problem.level.9.cannons       = true
eval.problem.level.9.blocks        = false
eval.problem.level.9.tubes         = false

```

## F Full Source Code

### F.1 Agent Framework

```
package com.montywest.marioai.rules

import ch.idisia.benchmark.mario.environments.Environment
import ch.idisia.benchmark.mario.engine.GeneralizerLevelScene
import Math.{min, max}
import scala.language.implicitConversions
import scala.language.postfixOps

import com.montywest.marioai.util.PrintUtils

abstract sealed class Perception(val index: Int) {
  def apply(environment: Environment): Byte
  def unapply(index: Int): Boolean = index == this.index
}

/**
 * An Byte Perception returns an Byte between 0 (inclusive) and limit (inclusive)
 */
abstract sealed class BytePerception(index: Int, val limit: Byte) extends Perception(index)

/**
 * A Bool Perception returns either true or false
 */
abstract sealed class BoolPerception(index: Int) extends Perception(index) {
  val TRUE: Byte = 1
  val FALSE: Byte = 0
}

/**
 * Perceives Mario's current mode.
 * Mario's mode is reduced by colliding with enemies or collecting green mushrooms.
 * It can be raised to BIG by a mushroom and FIRE by a fire flower.
 */
case object MarioMode extends BytePerception(0, 2) {
  val SMALL: Byte = 0; val LARGE: Byte = 1; val FIRE: Byte = 2;

  def apply(environment: Environment): Byte = {
    min( max(environment.getMarioMode(), 0), limit).toByte
  }
}

/**
 * Perceives if Mario can Jump.
 * This is true if the jump key is not pressed and Mario is on ground or wall sliding.
 */
case object JumpAvailable extends BoolPerception(1) {
  def apply(environment: Environment): Byte = {
    if(environment.isMarioAbleToJump) 1 else 0
  }
}

/**
 * Perceives whether or not Mario is standing on the ground
 */
case object OnGround extends BoolPerception(2) {
  def apply(environment: Environment): Byte = {
    if(environment.isMarioOnGround) 1 else 0
  }
}

/**
 * Perceives whether or not there is an enemy to Mario's left.
 * The perception works with in the declared box coordinates.
 */
case object EnemyLeft extends BoolPerception(3) {
  val AREA_UL = (-2,-2); val AREA_BR = (1, -1);

  def apply(environment: Environment): Byte = {
    if(Perception.enemyInBoxRelativeToMario(environment, AREA_UL, AREA_BR))
      1 else 0
  }
}

/**
 * Perceives whether or not there is an enemy to Mario's upper right.
 * The perception works with in the two declared box coordinates.
 */

```

```

case object EnemyUpperRight extends BoolPerception(4) {
  val AREA_1_UL = (-3, 0); val AREA_1_BR = (-1, 3);
  val AREA_2_UL = (-3, 3); val AREA_2_BR = (-2,5);

  def apply(environment: Environment): Byte = {
    if (Perception.enemyInBoxRelativeToMario(environment, AREA_1_UL, AREA_1_BR) ||
        Perception.enemyInBoxRelativeToMario(environment, AREA_2_UL, AREA_2_BR))
      1 else 0
  }
}

/**
 * Perceives whether or not there is an enemy to Mario's lower right.
 * The perception works with in the two declared box coordinates.
 */
case object EnemyLowerRight extends BoolPerception(5) {
  val AREA_1_UL = (0, 1); val AREA_1_BR = (1, 2);
  val AREA_2_UL = (-1, 2); val AREA_2_BR = (2, 2);

  def apply(environment: Environment): Byte = {
    if (Perception.enemyInBoxRelativeToMario(environment, AREA_1_UL, AREA_1_BR) ||
        Perception.enemyInBoxRelativeToMario(environment, AREA_2_UL, AREA_2_BR))
      1 else 0
  }
}

/**
 * Perceives whether or not there is an obstacle ahead of Mario.
 * It checks for rises in terrain, or steps in terrain in two boxes.
 * If there is a pit this perception is nullified.
 */
case object ObstacleAhead extends BoolPerception(6) {
  val AREA_OBS_UL = (-2, 1); val AREA_OBS_BR = (0, 3); //4
  val AREA_STEP_UL = (1, 1); val AREA_STEP_BR = (2, 3); //5

  def apply(environment: Environment): Byte = {
    if (Perception.obstacleInBoxRelativeToMario(environment, AREA_OBS_UL, AREA_OBS_BR) ||
        Perception.stepInBoxRelativeToMario(environment, AREA_STEP_UL, AREA_STEP_BR))
      1 else 0
  }
}

/**
 * Perceives whether or not there is a pit ahead of Mario.
 * Returns NONE if there are no pits, FAR and CLOSE is there is one in a certain number of columns relative to
 * mario.
 */
case object PitAhead extends BytePerception(7, 2) {
  val NONE: Byte = 0; val CLOSE: Byte = 1; val FAR: Byte = 2;
  val COL_CLOSE_L = 1; val COL_CLOSE_R = 1; val COL_FAR_L = 2; val COL_FAR_R = 2

  def apply(environment: Environment): Byte = {
    val pitUp: Option[Int] = Perception.getOpens(environment, COL_CLOSE_L, COL_FAR_R).headOption
    pitUp match {
      case None => NONE
      case Some(x) if (x >= COL_CLOSE_L && x <= COL_CLOSE_R) => CLOSE
      case Some(x) if (x >= COL_FAR_L && x <= COL_FAR_R) => FAR
      case _ => NONE
    }
  }
}

/**
 * Perceives whether or not there is a pit directly below Mario.
 */
case object PitBelow extends BoolPerception(8) {
  val COL_L = 0; val COL_R = 0;

  def apply(environment: Environment): Byte = {
    if (Perception.pitRelativeToMario(environment, COL_L, COL_R))
      1 else 0
  }
}

/**
 * Perceives the direction Mario is moving on the horizontal plane.
 */
case object MovingX extends BytePerception(9, 2) {
  val STILL: Byte = 0; val LEFT: Byte = 1; val RIGHT: Byte = 2;

  def apply(environment: Environment): Byte = {
    environment.getMarioMovement.apply(0) match {

```

```

        case 0 => {
            0
        }
        case -1 => {
            1
        }
        case 1 => 2
        case _ => 0
    }
}

/**
 * Perceives the direction Mario is moving on the vertical plane.
 */
case object MovingY extends BytePerception(10, 2) {
    val STILL: Byte = 0; val DOWN: Byte = 1; val UP: Byte = 2;

    def apply(environment: Environment): Byte = {
        environment.getMarioMovement.apply(1) match {
            case 0 => 0
            case 1 => 1
            case -1 => 2
            case _ => 0
        }
    }
}

object Perception {

    val NUMBER_OF_PERCEPTIONS = 11;

    implicit def per2int(perception: Perception): Int = perception.index

    /**
     * Extractor object, matches the perception on index.
     * Allows for code:
     * n: Int match {
     *   case Perception(perception) => perception...
     *   ...
     * }
     */
    def unapply(n: Int): Option[Perception] = n match {
        case MarioMode() => Option(MarioMode)
        case JumpAvailable() => Option(JumpAvailable)
        case OnGround() => Option(OnGround)
        case EnemyLeft() => Option(EnemyLeft)
        case EnemyUpperRight() => Option(EnemyUpperRight)
        case EnemyLowerRight() => Option(EnemyLowerRight)
        case ObstacleAhead() => Option(ObstacleAhead)
        case PitAhead() => Option(PitAhead)
        case PitBelow() => Option(PitBelow)
        case MovingX() => Option(MovingX)
        case MovingY() => Option(MovingY)
        case _ => None
    }

    val EGO_POS_ROW_INDEX = 0;
    val EGO_POS_COL_INDEX = 1;

    /**
     * Checks if there is an enemy in a box relative to Mario.
     */
    def enemyInBoxRelativeToMario(environment: Environment, a: (Int, Int), b: (Int, Int)): Boolean = {
        val enemies = environment.getEnemiesObservationZ(2); //Z-index 2 gives 1 for enemy, 0 for anything else
        val test = (grid: Array[Array[Byte]], tup: Tuple2[Int, Int]) => {
            val x = grid(tup._1)(tup._2)
            x == 1
        }

        checkBox(enemies, test, getMarioPos(environment), a, b)
    }

    /**
     * Checks if there is an obstacle (rise in terrain) in a box relative to Mario.
     */
    def obstacleInBoxRelativeToMario(environment: Environment, a: (Int, Int), b: (Int, Int)): Boolean = {
        val level = environment.getLevelSceneObservationZ(2); //
        val test = (grid: Array[Array[Byte]], tup: Tuple2[Int, Int]) => {
            val x = grid(tup._1)(tup._2)
            obs(x)
        }
    }
}

```

```

val opens = this.getOpens(environment, a._2, b._2);

val b2 = (b._1, {
  opens.headOption match {
    case None => b._2
    case Some(x) => x-1
  }
})

if (b2._2 < a._2) {
  false
} else {
  checkBox(level, test, getMarioPos(environment), a, b2)
}
}

/**
 * Checks if there is an step (fall then rise in terrain) in a box relative to Mario.
 */
def stepInBoxRelativeToMario(environment: Environment, a: (Int, Int), b: (Int, Int)): Boolean = {
  val level = environment.getLevelSceneObservationZ(2); //
  val test = (grid: Array[Array[Byte]], tup: Tuple2[Int, Int]) => {
    val y = tup._2; val x = tup._1;
    (y != Math.max(a._2, b._2)) && obs(grid(x)(y+1)) && !obs(grid(x)(y))
  }

  val opens = this.getOpens(environment, a._2, b._2);
  val b2 = (b._1, {
    opens.headOption match {
      case None => b._2
      case Some(x) => x-1
    }
  })

  if (b2._2 < a._2) {
    false
  } else {
    checkBox(level, test, getMarioPos(environment), a, b2)
  }
}

/**
 * Checks whether a box is filled with terrain, realtive to Mario
 */
def obstacleFillBoxRelativeToMario(environment: Environment, a: (Int, Int), b: (Int, Int)): Boolean = {
  val level = environment.getLevelSceneObservationZ(2); //
  val test = (grid: Array[Array[Byte]], tup: Tuple2[Int, Int]) => {
    val x = grid(tup._1)(tup._2)
    !obs(x)
  }

  checkBox(level, test, getMarioPos(environment), a, b, false)
}

/**
 * Checks if there is an pit in columns relative to Mario.
 */
def pitRelativeToMario(environment: Environment, a: Int, b: Int): Boolean = {
  val opens = this.getOpens(environment, a, b)
  !opens.isEmpty
}

/**
 * Helper function for box checks
 */
private def checkBox(grid: Array[Array[Byte]], test: (Array[Array[Byte]], (Int, Int))=>Boolean, mario: (Int, Int), a: (Int, Int), b: (Int, Int), ret: Boolean = true): Boolean = {
  import Math.min
  import Math.max

  val relARow = min(grid.length-1, max(0, (a._1 + mario._1)))
  val relACol = min(grid(0).length-1, max(0, (a._2 + mario._2)))
  val relBRow = min(grid.length-1, max(0, (b._1 + mario._1)))
  val relBCol = min(grid(0).length-1, max(0, (b._2 + mario._2)))

  for {
    i <- min(relARow, relBRow) to max(relARow, relBRow)
    j <- min(relACol, relBCol) to max(relARow, relBCol)
    if (test(grid, (i, j)))
  }{
    return ret
  }
  !ret
}

```

```

}

/**
 * Gets open columns below Mario
 */
def getOpens(environment: Environment, a: Int, b: Int): List[Int] = {
  val level = environment.getLevelSceneObservationZ(2); //
  val test = (x: Byte) => x == 0 || x == GeneralizerLevelScene.COIN_ANIM;
  val bottomRow = level.length - 1
  val mario = getMarioPos(environment)

  import Math.min
  import Math.max

  val left = max(0, min(a, b) + mario._2)
  val right = min(level(0).length, max(a, b) + mario._2)

  var opens: List[Int] = left to right toList

  for {
    i <- mario._1 + 1 to bottomRow
    if (!opens.isEmpty)
  }{
    opens = opens.filter { j => level(i)(j) == 0 || level(i)(j) == GeneralizerLevelScene.COIN_ANIM }
  }
  opens.map { x => x - mario._2 }
}

/**
 * Byte value for terrain
 */
private def obs(b: Byte): Boolean = {
  b == 1 || b == GeneralizerLevelScene.BORDER_CANNOT_PASS_THROUGH
}

/**
 * Gets Mario's position on scene.
 */
private def getMarioPos(environment: Environment): (Int, Int) = {
  val marioPos = environment.getMarioEgoPos;
  (marioPos(EGO_POS_ROW_INDEX), marioPos(EGO_POS_COL_INDEX))
}
}

```

---

```

package com.montywest.marioai.rules

sealed trait KeyPress;

case object KeyLeft extends KeyPress;
case object KeyRight extends KeyPress;
case object KeyJump extends KeyPress;
case object KeySpeed extends KeyPress;

```

---

```

package com.montywest.marioai

import scala.language.implicitConversions

package object rules {
  type Observation = Vector[Byte]
  object Observation {
    import ch.idisia.benchmark.mario.environments.Environment

    def apply(environment: Environment): Observation = {
      Vector.tabulate(Perception.NUMBER_OF_PERCEPTIONS)
        ({n: Int => n match {
          case Perception(perception) => perception(environment)
          case _ => throw new IllegalArgumentException
        }})
    }
  }

  val BLANK_OBSERVATION: Observation = Vector.fill(Perception.NUMBER_OF_PERCEPTIONS)(0)
}

type Conditions = Vector[Byte]
object Conditions {
  val DONT_CARE: Byte = -1;
}

```



```

val LENGTH = Perception.NUMBER_OF_PERCEPTIONS

def validateCondition(cond: (Perception, Byte)): Boolean = cond match {
  case (bp : BoolPerception, b) => (bp.TRUE == b) || (bp.FALSE == b)
  case (ip : BytePerception, b) => (0 <= b) && (b <= ip.limit)
}

def validateConditions(conditionsVector: Vector[Byte]): Boolean = {
  conditionsVector.zipWithIndex.forall {
    case (b, Perception(perception)) => perception match {
      case boolP : BoolPerception => (b == boolP.TRUE) || (b == boolP.FALSE) || (b == DONT_CARE)
      case byteP : BytePerception => ((0 <= b) && (b <= byteP.limit)) || (b == DONT_CARE)
    }
    case _ => false
  }
}

def getLimitForIndex(index: Int): Byte = index match {
  case Perception(perception) => perception match {
    case bp : BoolPerception => Math.max(bp.TRUE, bp.FALSE).toByte
    case ip : BytePerception => ip.limit
  }
  case _ => throw new IllegalArgumentException
}

def apply(conditions: Map[Perception, Byte]): Conditions = {
  if (conditions.forall(validateCondition)) {
    val perMap = conditions.map { case (p, b) => (p.index, b) }
    Vector.tabulate(Conditions.LENGTH)(i: Int) =>
      perMap.get(i) match {
        case None => Conditions.DONT_CARE
        case Some(b: Byte) => b
      }
  } else throw new IllegalArgumentException("Perception byte value out of range")
}

type ExAction = Array[Boolean]
object ExAction {
  import ch.idsia.benchmark.mario.environments.Environment
  import ch.idsia.benchmark.mario.engine.sprites.Mario

  val KEY_LEFT = Mario.KEY_LEFT
  val KEY_RIGHT = Mario.KEY_RIGHT
  val KEY_UP = Mario.KEY_UP
  val KEY_DOWN = Mario.KEY_DOWN
  val KEY_JUMP = Mario.KEY_JUMP
  val KEY_SPEED = Mario.KEY_SPEED

  val LENGTH = Environment.numberOfKeys

  def apply(keys: KeyPress*): ExAction = {
    val ac: ExAction = Array.fill(LENGTH)(false)
    keys.foreach {
      case KeyLeft => ac(KEY_LEFT) = true;
      case KeyRight => ac(KEY_RIGHT) = true;
      case KeyJump => ac(KEY_JUMP) = true;
      case KeySpeed => ac(KEY_SPEED) = true;
    }
    ac
  }

  def build(keys: Set[KeyPress]): ExAction = {
    Array.tabulate(LENGTH)(i: Int) => i match {
      case KEY_LEFT => keys.contains(KeyLeft)
      case KEY_RIGHT => keys.contains(KeyRight)
      case KEY_JUMP => keys.contains(KeyJump)
      case KEY_SPEED => keys.contains(KeySpeed)
      case _ => false
    }
  }

  implicit def apply(mwAction: MWAction): ExAction = {
    Array.tabulate(LENGTH)(x: Int) =>
      x match {
        case KEY_LEFT => MWAction.ACTION_TRUE == mwAction(MWAction.LEFT_INDEX)
        case KEY_RIGHT => MWAction.ACTION_TRUE == mwAction(MWAction.RIGHT_INDEX)
        case KEY_JUMP => MWAction.ACTION_TRUE == mwAction(MWAction.JUMP_INDEX)
        case KEY_SPEED => MWAction.ACTION_TRUE == mwAction(MWAction.SPEED_INDEX)
        case _ => false
      }
  }
}

```

```

    )
  }
}

type MWAAction = Vector[Byte]
object MWAAction {
  val LENGTH = 4
  val LEFT_INDEX = 0;
  val RIGHT_INDEX = 1;
  val JUMP_INDEX = 2;
  val SPEED_INDEX = 3;

  val ACTION_TRUE: Byte = 1;
  val ACTION_FALSE: Byte = 0;

  def apply(keys: KeyPress*): MWAAction = {
    build(keys.toSet)
  }

  def build(keys: Set[KeyPress]): MWAAction = {
    val actIndexSet = keys.map { getKeyPressIndex }

    Vector.tabulate(LENGTH)( (i: Int) =>
      if (actIndexSet.contains(i)) ACTION_TRUE
      else ACTION_FALSE
    )
  }

  def build(left: Boolean, right: Boolean, jump: Boolean, speed: Boolean): MWAAction = {
    Vector.tabulate(LENGTH)((i: Int) => {
      if (i match {
        case LEFT_INDEX => left
        case RIGHT_INDEX => right
        case JUMP_INDEX => jump
        case SPEED_INDEX => speed
      })
      {ACTION_TRUE}
      else
      {ACTION_FALSE}
    })
  }

  def getKeyPressIndex(kp: KeyPress): Int = kp match {
    case KeyLeft => LEFT_INDEX
    case KeyRight => RIGHT_INDEX
    case KeyJump => JUMP_INDEX
    case KeySpeed => SPEED_INDEX
  }
}
}

```

---

```

package com.montywest.marioai.rules

import scala.annotation.migration
import scala.annotation.tailrec

class Rule private[Rule] (vector: Vector[Byte]) {

  def getVectorRep: Vector[Byte] = {
    vector
  }

  def getConditions: Conditions = {
    vector.slice(0, Conditions.LENGTH)
  }

  def getExAction: ExAction = {
    ExAction(getMWAAction)
  }

  def getMWAAction: MWAAction = {
    vector.slice(Conditions.LENGTH, Rule.TOTAL_LENGTH)
  }

  def scoreAgainst(observation: Observation): Int = {
    if (observation.length != Conditions.LENGTH)
      throw new IllegalArgumentException("observation is a different length to conditions")
    @tailrec
    def scoreRecu(i: Int, sum: Int = 0): Int = {
      if (i == Conditions.LENGTH) sum
      else getConditions(i) match {

```

```

        case Conditions.DONT_CARE => scoreRecu(i+1, sum)
        case b if b == observation(i) => scoreRecu(i+1, sum+1)
        case _ => -1
    }
}
scoreRecu(0)
}

override def toString(): String = {
    val b2str = (b: Byte) => b match {
        case -1 => "_"
        case b => b.toString()
    }

    (getConditions.map { b2str } mkString(" ") + " | " + (getMWAction map { _.toString } mkString(" ")))
}

override def hashCode: Int =
    this.getVectorRep.hashCode()

override def equals(other: Any): Boolean = other match {
    case that: Rule =>
        that.getVectorRep == this.getVectorRep
    case _ => false
}

}

object Rule {

    val PRINT_HEADER = "MM JA OG EL UR LR OA PA PB MX MY| KL KR KJ KS"

    val TOTAL_LENGTH = Conditions.LENGTH + MWAction.LENGTH

    //Creationnal
    private val ACTION_LEFT_INDEX = MWAction.LEFT_INDEX + Conditions.LENGTH;
    private val ACTION_RIGHT_INDEX = MWAction.RIGHT_INDEX + Conditions.LENGTH;
    private val ACTION_JUMP_INDEX = MWAction.JUMP_INDEX + Conditions.LENGTH;
    private val ACTION_SPEED_INDEX = MWAction.SPEED_INDEX + Conditions.LENGTH;

    val BLANK_RULE = new Rule(
        Vector.fill(Conditions.LENGTH)(Conditions.DONT_CARE) ++ Vector.fill(MWAction.LENGTH)(MWAction.ACTION_FALSE)
    )

    def apply(vec: Vector[Byte]): Rule = {
        if (vec.length == Conditions.LENGTH + MWAction.LENGTH)
            new Rule(vec)
        else
            throw new IllegalArgumentException("Rule vector length incorrect")
    }

    def build(conditionMap: Map[Perception, Byte], actionSet: Set[KeyPress]): Rule = {
        new Rule(Conditions(conditionMap) ++ MWAction.build(actionSet))
    }

    def apply(conditions: Conditions, action: MWAction): Rule = {
        if ((conditions.length == Conditions.LENGTH) && (action.length == MWAction.LENGTH)) {
            new Rule(conditions ++ action);
        } else {
            throw new IllegalArgumentException("Rule vector length incorrect")
        }
    }
}

}

package com.montywest.marioai.rules

import scala.annotation.tailrec
import scala.language.implicitConversions
import scala.language.postfixOps
import scala.collection.mutable.WrappedArray
import scala.collection.mutable.Map

/**
 * Container for an agent's rules.
 * Includes a default action if no rules match the current environment
 */
class Ruleset( val rules: Seq[Rule], val defaultAction: MWAction, favourHigher: Boolean = true) {

    /**

```

```

    * Logs which rules are used and how many times.
    */
    val ruleUsage: Map[Int, Int] = Map();

    def length = rules.length

    /**
     * Returns true is second argument is strictly higher than first argument.
     * If they are equal then result return is NOT favourHigher.
     */
    private val newRuleBetter: (Int, Int) => Boolean = {
        if(favourHigher)
            (best:Int, newScore: Int) => best<newScore
        else
            (best:Int, newScore: Int) => best<=newScore
    }

    /**
     * Applies observation ot rule set
     * Returns the action that belongs the highest scoring rule,
     * as an ExAction.
     */
    def getBestExAction(observation: Observation): ExAction = {

        @tailrec
        def getBestRuleRecu(ls: Seq[Rule], best: Option[(Int, Rule)] = None, bestScore: Int = 0, index: Int = 0):
            Option[(Int, Rule)] = ls match {
                case Nil => best
                case (r +: ts) => {
                    val newScore = r.scoreAgainst(observation)
                    if (newRuleBetter(bestScore, newScore))
                        getBestRuleRecu(ts, Some((index, r)), newScore, index+1)
                    else
                        getBestRuleRecu(ts, best, bestScore, index+1)
                }
            }
        getBestRuleRecu(rules) match {
            case None => {
                this.incrementRuleUsage(-1)
                ExAction(defaultAction)
            }
            case Some((i,r)) => {
                this.incrementRuleUsage(i)
                r.getExAction
            }
        }
    }

    /**
     * Returns the vector representation of the ruleset,
     * defaultAction can be removed with parameter.
     * If included it is represented by rule contain all DONT_CARE for conditions
     */
    def getVectorRep(withDefaultAction: Boolean = true): Vector[Byte] = {
        if (!withDefaultAction)
            (rules.map { r => r.getVectorRep } toVector).flatten
        else
            (rules.map { r => r.getVectorRep } toVector).flatten ++
            Vector.fill(Conditions.LENGTH)(-1: Byte) ++ defaultAction
    }

    /**
     * Increments the map store of rule usage for rule index
     */
    private def incrementRuleUsage(index: Int): Unit = {
        // println("Rule used: " + index)
        ruleUsage.get(index) match {
            case None => ruleUsage.put(index, 1)
            case Some(x) => ruleUsage.put(index, x+1)
        }
    }

    def resetRuleUsage = {
        ruleUsage.clear()
    }

    override def toString: String = {
        rules.zipWithIndex map {
            case (r:Rule, i:Int) => {
                val sep = if (i<10) " " else ""
                sep + i + ". " + r.toString()
            }
        } mkString(

```

```

        " " + Rule.PRINT_HEADER + "\n",
        "\n",
        "\n-1. " + Rule(Vector.fill(Conditions.LENGTH)(-1: Byte) ++ defaultAction).toString()
    }

    override def hashCode: Int =
        this.getVectorRep(true).hashCode()

    override def equals(other: Any): Boolean = other match {
        case that: Rule =>
            that.getVectorRep == this.getVectorRep(true)
        case _ => false
    }
}

object Ruleset {

    val FALLBACK_ACTION = MWAction(KeyRight, KeySpeed, KeyJump)

    /**
     * Builds a ruleset from a set of rules and default action keypresses
     */
    def apply(rules: Seq[Rule], defaultAction: Set[KeyPress]): Ruleset = {
        new Ruleset(rules, MWAction.build(defaultAction))
    }

    /**
     * Builds a ruleset from its vector representation.
     * If the vector contains a DONT_CARE rule as its final rule, then this is used as default action
     * A fallback default action can be passed that is used if a default action is not present.
     */
    def build(vec: Vector[Byte], fallbackDefaultAction: MWAction = FALLBACK_ACTION): Ruleset = {
        val twodim = vec.grouped(Rule.TOTAL_LENGTH).toVector
        val lastIndex = twodim.length - 1
        if (twodim(lastIndex).length == Rule.TOTAL_LENGTH) {
            val lastIndexSplit = twodim(lastIndex).splitAt(Conditions.LENGTH)
            //Default Included
            if (lastIndexSplit._1.forall { _ == (-1: Byte) }) {
                new Ruleset(twodim.slice(0, lastIndex) map { vr => Rule(vr) } toSeq,
                    lastIndexSplit._2)
            }
            // No Default
            else {
                new Ruleset(twodim map { vr => Rule(vr) } toSeq,
                    fallbackDefaultAction)
            }
        } else throw new IllegalArgumentException("Malformed vector representation of ruleset")
    }

    /**
     * Builds a ruleset from its array representation.
     * If the array contains a DONT_CARE rule as its final rule, then this is used as default action
     * A fallback default action can be passed that is used if a default action is not present.
     */
    def buildFromArray(arr: Array[Byte], fallbackAction: MWAction): Ruleset = {
        val wa: WrappedArray[Byte] = arr
        Ruleset.build(wa.toVector, fallbackAction)
    }
}

}

package com.montywest.marioai.agents

import com.montywest.marioai.rules.Observation

import ch.idsia.agents.Agent
import ch.idsia.agents.controllers.human.HumanKeyboardAgent
import ch.idsia.benchmark.mario.environments.Environment

class MWHumanAgent extends HumanKeyboardAgent with Agent {

    var observation: Observation = Observation.BLANK_OBSERVATION

    override def integrateObservation(env: Environment) {
        super.integrateObservation(env)
        observation = Observation(env)
    }
}

```

---

```

package com.montywest.marioai.agents

import com.montywest.marioai.rules._

import ch.idisia.agents.Agent
import com.montywest.marioai.rules.Observation
import ch.idisia.benchmark.mario.environments.Environment

abstract class MWObservationAgent extends Agent {

  var observation: Observation = Observation.BLANK_OBSERVATION

  override def integrateObservation(environment: Environment) = {
    try {
      observation = Observation(environment);
    } catch {
      case e: Exception => println(e.getStackTrace); throw e
    }
  }

  override def giveIntermediateReward(reward: Float) = {}

  override def reset = {};

  override def setObservationDetails(rfWidth: Int, rfHeight: Int, egoRow: Int, egoCol: Int) = {}
}

```

---

```

package com.montywest.marioai.agents

import ch.idisia.agents.Agent
import com.montywest.marioai.rules.Ruleset
import com.montywest.marioai.rules.ExAction

class MWRulesetAgent(var name: String, val ruleset: Ruleset) extends MWObservationAgent with Agent {

  def getName = name
  def setName(name: String) = this.name = name

  override def getAction: ExAction = {
    ruleset.getBestExAction(observation)
  }

  def resetRuleUsage = {
    ruleset.resetRuleUsage
  }
}

object MWRulesetAgent {
  def apply(name: String, ruleset: Ruleset): MWRulesetAgent = {
    new MWRulesetAgent(name, ruleset)
  }
}

```

---

```

package com.montywest.marioai.agents

import ch.idisia.agents.Agent
import java.io.FileNotFoundException
import com.montywest.marioai.rules.Ruleset
import com.montywest.marioai.rules.Ruleset
import scala.io.BufferedSource
import com.montywest.marioai.rules.Ruleset
import com.montywest.marioai.rules.Ruleset
import java.util.regex.Pattern
import java.io.FileWriter
import java.io.IOException
import com.montywest.marioai.rules.Rule
import com.montywest.marioai.rules.Conditions
import java.io.File
import scala.language.postfixOps

object MWRulesetAgentIO {

  private val postfix = ".agent"
  private val writeFolder = "agents/written/"

```

```

val CSV_HEADER = "cMM,cJA,cOG,cEL,cEUR,cELR,cOA,cPA,cPB,cMX,cMY,aL,aR,aJ,aS"

def fromFile(filename: String, lineStart: Int = 0): MWRulesetAgent = {
  var bufferedSource: Option[BufferedSource] = None
  try {
    bufferedSource = Some(io.Source.fromFile(filename))
    val lines = bufferedSource.get.getLines().toVector.drop(lineStart)
    val letters = "[a-zA-Z]".r
    val linesDrop =
      if (lines.headOption.isDefined && letters.findFirstIn(lines.head).isDefined) {
        lines.drop(1)
      } else
        lines

    val vec: Vector[Byte] = linesDrop.toStream.takeWhile { (s: String) => !s.trim.isEmpty() }.flatMap {
      (s: String) => s.split(",").map { _.trim.toByte }
    }.toVector;

    val filenameArr = filename.split(Pattern.quote("/"))
    new MWRulesetAgent( filenameArr.last.replace(".agent", ""), Ruleset.build(vec))
  } catch {
    case e: FileNotFoundException => throw new IllegalArgumentException("Could not find agent file.", e)
    case e: NumberFormatException => throw new IllegalArgumentException("Non-number found in agent file (after
      header).", e)
  } finally {
    if (bufferedSource.isDefined) bufferedSource.get.close
  }
}

def toFile(filename: String, agent: MWRulesetAgent, header: Boolean): Unit = {
  val filenameFull = if(!filename.contains(".")) {
    filename + postfix
  } else filename

  val rules = agent.ruleset.rules
  val defaultA = agent.ruleset.defaultAction

  var writerOpt: Option[FileWriter] = None;
  try {
    val dir: File = new File(writeFolder);
    if (!dir.exists()) { dir.mkdirs() }

    writerOpt = Some(new FileWriter(writeFolder + filenameFull))

    val writer = writerOpt.get
    if(header) writer.append(CSV_HEADER + "\n")
    rules.foreach { (r: Rule) => {
      writer.append(r.getVectorRep.mkString(", "))
      writer.append("\n")
    }}
    writer.append((Vector.fill(Conditions.LENGTH)(-1:Byte) ++ defaultA).mkString(", "))
    writer.append("\n")

    writer.append("\n\n")
    writer.flush()
  } catch {
    case e: IOException => throw new IllegalArgumentException("File inaccessible or is a folder, or error on
      flush", e)
  } finally {
    if (writerOpt.isDefined) writerOpt.get.close
  }
}

}

}



---



package com.montywest.marioai.util

object PrintUtils {

  def sceneToString(scene: Array[Array[Byte]], markCenter: Boolean = false): String = {
    if (markCenter) scene(scene.length/2)(scene(0).length/2) = 8
    scene.map {a: Array[Byte] => a.mkString(" ") } mkString("\n")
  }

}



---



package com.montywest.marioai.util

```

```

import scala.util.Random

object RandomUtils {

  private val r = new Random(System.currentTimeMillis)

  def randomIntEx(lowerIn: Int, higherEx: Int): Int = {
    r.nextInt(highestEx - lowerIn) + lowerIn
  }
  def randomIntIn(lowerIn: Int, higherIn: Int): Int = {
    r.nextInt(highestIn+1 - lowerIn) + lowerIn
  }

  def randomByteEx(lowerIn: Byte, higherEx: Byte): Byte = {
    (r.nextInt(highestEx.toInt - lowerIn.toInt) + lowerIn.toInt).toByte
  }
  def randomByteIn(lowerIn: Byte, higherIn: Byte): Byte = {
    (r.nextInt(highestIn.toInt+1 - lowerIn.toInt) + lowerIn.toInt).toByte
  }

  def randomIntTupleInclusive(lowerIn: (Int, Int), higherIn: (Int, Int)): (Int, Int) = {
    (lowerIn, higherIn) match {
      case ((x1, y1), (x2, y2)) if x1 <= x2 && y1 <= y2 =>
        (randomIntIn(x1, x2), randomIntIn(y1, y2))
      case _ => throw new IllegalArgumentException
    }
  }
}

package com.montywest.marioai.fixtures

object EnvironmentFixtures {

  val mario = (9, 9)
  val sceneSize = (19, 19)

  def baseLevelScene(groundLevel: Int): Array[Array[Byte]] = Array.fill(sceneSize._1, sceneSize._2)(0: Byte)
    .updated(groundLevel+mario._1 + 1, Array.fill(sceneSize._2)(1: Byte))
    .updated(groundLevel+mario._1, Array.fill(sceneSize._2)(1: Byte));

  val baseEnemies: Array[Array[Byte]] = Array.fill(sceneSize._1, sceneSize._2)(0: Byte);

  def getEnemyScene(enemy: (Int, Int)): Array[Array[Byte]] = {
    Array.tabulate(sceneSize._1, sceneSize._2)((x: Int, y: Int) => {
      if (enemy == (x-mario._1, y-mario._2)) 1 else 0
    })
  }

  def getEnemyScene(enemies: Set[(Int, Int)]): Array[Array[Byte]] = {
    Array.tabulate(sceneSize._1, sceneSize._2)((x: Int, y: Int) => if (enemies.contains(x-mario._1, y-mario._2))
      1 else 0)
  }

  def getLevelSceneWithPit(groundLevel: Int, pitCols: Set[Int]): Array[Array[Byte]] = {
    Array.tabulate(sceneSize._1, sceneSize._2)(
      (x: Int, y: Int) => {
        if (pitCols.contains(y-mario._2)) 0
        else if (x-mario._1 == groundLevel) 1
        else 0
      })
  }

  def getCustomLevelScene(filledRows: Set[Int], removals: Set[(Int,Int)], additions: Set[(Int, Int)]): Array[
    Array[Byte]] = {
    Array.tabulate(sceneSize._1, sceneSize._2)(
      (x: Int, y: Int) => {
        if (removals.contains((x-mario._1, y-mario._2))) 0
        else if (additions.contains((x-mario._1, y-mario._2))) 1
        else if (filledRows.contains(x-mario._1)) 1
        else 0
      })
  }
}

package com.montywest.marioai.fixtures

import com.montywest.marioai.rules.Observation
import com.montywest.marioai.rules.Perception

```



```

object RulePackageFixtures {

  def getObservation(obs: Map[Perception, Byte]): Observation = {
    val map = obs.map { case (p, b) => (p.index, b) }
    Vector.tabulate(Perception.NUMBER_OF_PERCEPTIONS){ (i: Int) =>
      map.get(i) match {
        case None => 0
        case Some(b: Byte) => b
      }
    }
  }
}

package com.montywest.marioai.rules

import org.scalamock.scalatest.MockFactory
import org.scalatest.BeforeAndAfterAll
import org.scalatest.FlatSpec
import org.scalatest.Matchers

class ConditionsTest extends FlatSpec with Matchers with MockFactory with BeforeAndAfterAll {

  "validateCondition" should "should return true if condition is within perception range" in {

    assert(Conditions.validateCondition((MarioMode, MarioMode.LARGE)))
    assert(Conditions.validateCondition((OnGround, 0: Byte)))
    assert(Conditions.validateCondition((EnemyLeft, 1: Byte)))
    assert(Conditions.validateCondition((MovingX, MovingX.RIGHT)))

  }

  "validateCondition" should "should return false if condition is outside perception range" in {

    assert(!Conditions.validateCondition((EnemyUpperRight, -2: Byte)))
    assert(!Conditions.validateCondition((MovingY, (MovingY.limit + 5).toByte)))

  }

  "apply" should "turn a map in Conditions (with DONT_CARE as default)" in {

    val vec: Vector[Byte] = Vector.tabulate(Conditions.LENGTH){ _ match {
      case MarioMode() => 2
      case OnGround() => 1
      case ObstacleAhead() => 1
      case MovingX() => 2
      case _ => Conditions.DONT_CARE
    }}

    assertResult(vec) {
      Conditions(Map(MarioMode -> 2,
        OnGround -> 1,
        ObstacleAhead -> 1,
        MovingX -> 2))
    }

  }

  "apply" should "throw exception if passed a perception out of range" in {

    intercept[Exception] {
      Conditions(Map(MarioMode -> 2,
        OnGround -> 1,
        ObstacleAhead -> 1,
        MovingX -> (MovingX.limit + 5).toByte))
    }

  }
}

package com.montywest.marioai.rules

import org.scalatest.BeforeAndAfterAll
import org.scalamock.scalatest.MockFactory
import org.scalatest.Matchers
import org.scalatest.FlatSpec
import ch.idsia.benchmark.mario.environments.Environment
import ch.idsia.benchmark.mario.engine.sprites.Mario

class ExActionTest extends FlatSpec with Matchers with MockFactory with BeforeAndAfterAll {

```

```

"LENGTH" should "be equals number of keys in marioengine" in {
  assert(Environment.numberOfKeys == ExAction.LENGTH)
}

"ExAction indexes" should "match those in marioengine" in {
  assert(Environment.MARIO_KEY_DOWN == ExAction.KEY_DOWN)
  assert(Environment.MARIO_KEY_JUMP == ExAction.KEY_JUMP)
  assert(Environment.MARIO_KEY_RIGHT == ExAction.KEY_RIGHT)
  assert(Mario.KEY_UP == ExAction.KEY_UP)
  assert(Environment.MARIO_KEY_SPEED == ExAction.KEY_SPEED)
  assert(Environment.MARIO_KEY_LEFT == ExAction.KEY_LEFT)
}

"apply" should "take KeyPress array and return ExAction" in {
  val arr: Array[Boolean] = Array.tabulate(ExAction.LENGTH)(_ match {
    case ExAction.KEY_LEFT => true
    case ExAction.KEY_JUMP => true
    case _ => false
  })

  assertResult(arr.deep) {
    ExAction(KeyLeft, KeyJump).deep
  }
}

"build" should "take KeyPress set and return ExAction" in {
  val arr: Array[Boolean] = Array.tabulate(ExAction.LENGTH)(_ match {
    case ExAction.KEY_LEFT => true
    case ExAction.KEY_JUMP => true
    case _ => false
  })

  assertResult(arr.deep) {
    ExAction.build(Set(KeyLeft, KeyJump)).deep
  }
}

"apply" should "take MWAAction and produce ExAction" in {
  val arr: Array[Boolean] = Array.tabulate(ExAction.LENGTH)(_ match {
    case ExAction.KEY_LEFT => true
    case ExAction.KEY_JUMP => true
    case _ => false
  })

  val mwa: MWAAction = Vector.tabulate(MWAAction.LENGTH)(_ match {
    case MWAAction.LEFT_INDEX => MWAAction.ACTION_TRUE
    case MWAAction.JUMP_INDEX => MWAAction.ACTION_TRUE
    case _ => MWAAction.ACTION_FALSE
  })

  assert(arr.deep == ExAction(mwa).deep)
}
}

```

---

```

package com.montywest.marioai.rules

```

```

import org.scalatest.BeforeAndAfterAll
import org.scalamock.scalatest.MockFactory
import org.scalatest.Matchers
import org.scalatest.FlatSpec

```

```

class MWAActionTest extends FlatSpec with Matchers with MockFactory with BeforeAndAfterAll {

  "getKeyPressIndex" should "return correct index for KeyPress" in {
    assert(MWAAction.LEFT_INDEX == MWAAction.getKeyPressIndex(KeyLeft))
    assert(MWAAction.RIGHT_INDEX == MWAAction.getKeyPressIndex(KeyRight))
    assert(MWAAction.JUMP_INDEX == MWAAction.getKeyPressIndex(KeyJump))
    assert(MWAAction.SPEED_INDEX == MWAAction.getKeyPressIndex(KeySpeed))
  }

  "apply" should "take KeyPress array and return MWAAction" in {
    val vec: Vector[Byte] = Vector.tabulate(MWAAction.LENGTH)(_ match {
      case MWAAction.LEFT_INDEX => MWAAction.ACTION_TRUE
      case MWAAction.JUMP_INDEX => MWAAction.ACTION_TRUE
      case _ => MWAAction.ACTION_FALSE
    })

    assertResult(vec) {
      MWAAction(KeyLeft, KeyJump)
    }
  }
}

```

```

"build" should "take KeyPress set and return MWACTION" in {
  val vec: Vector[Byte] = Vector.tabulate(MWACTION.LENGTH)(_ match {
    case MWACTION.RIGHT_INDEX => MWACTION.ACTION_TRUE
    case MWACTION.SPEED_INDEX => MWACTION.ACTION_TRUE
    case _ => MWACTION.ACTION_FALSE
  })

  assertResult(vec) {
    MWACTION.build(Set(KeyRight, KeySpeed))
  }
}
}

```

---

```

package com.montywest.marioai.rules

import org.scalatest.BeforeAndAfterAll
import org.scalamock.scalatest.MockFactory
import org.scalatest.Matchers
import org.scalatest.FlatSpec

class ObservationTest extends FlatSpec with Matchers with MockFactory with BeforeAndAfterAll {

}

```

---

```

package com.montywest.marioai.rules

import org.scalamock.scalatest.MockFactory
import org.scalatest.FlatSpec
import ch.idisia.benchmark.mario.environments.Environment
import org.scalatest.Matchers
import ch.idisia.benchmark.mario.engine.sprites.Mario
import com.montywest.marioai.fixtures.EnvironmentFixtures
import com.montywest.marioai.util.RandomUtils
import com.montywest.marioai.util.PrintUtils
import org.scalatest.BeforeAndAfterAll

class PerceptionTest extends FlatSpec with Matchers with MockFactory with BeforeAndAfterAll {

  info("Starting...")
  var envStub = stub[Environment]

  override def beforeAll {
    envStub = stub[Environment]
  }

  "MarioMode apply" should "return 0 for small Mario" in {

    (envStub.getMarioMode _) when() returns(0)

    assertResult(0: Byte) {
      MarioMode(envStub)
    }
    assert(MarioMode.SMALL == (0: Byte))
  }

  "MarioMode apply" should "return 1 for Large Mario" in {

    (envStub.getMarioMode _) when() returns(1)

    assertResult(1: Byte) {
      MarioMode(envStub)
    }
    assert(MarioMode.LARGE == (1: Byte))
  }

  "MarioMode apply" should "return 2 for FIRE Mario" in {

    (envStub.getMarioMode _) when() returns(2)

    assertResult(2: Byte) {
      MarioMode(envStub)
    }
    assert(MarioMode.FIRE == (2: Byte))
  }

  "JumpAvailable apply" should "return 1 for true" in {

```

```

        (envStub.isMarioAbleToJump _) when() returns(true)

        assertResult(1: Byte) {
            JumpAvailable(envStub)
        }
    }

    "JumpAvailable apply" should "return 0 for false" in {

        (envStub.isMarioAbleToJump _) when() returns(false)

        assertResult(0: Byte) {
            JumpAvailable(envStub)
        }
    }

    "OnGround apply" should "return 1 for true" in {

        (envStub.isMarioOnGround _) when() returns(true)

        assertResult(1: Byte) {
            OnGround(envStub)
        }
    }

    "OnGround apply" should "return 0 for false" in {

        (envStub.isMarioOnGround _) when() returns(false)

        assertResult(0: Byte) {
            OnGround(envStub)
        }
    }

    "MovingX apply" should "return 0 for not moving" in {
        (envStub.getMarioMovement _) when() returns(Array(0, 1))

        assertResult(0: Byte) {
            MovingX(envStub)
        }
        assert(MovingX.STILL == (0: Byte))
    }

    "MovingX apply" should "return 1 for moving left" in {
        (envStub.getMarioMovement _) when() returns(Array(-1, 0))

        assertResult(1: Byte) {
            MovingX(envStub)
        }
        assert(MovingX.LEFT == (1: Byte))
    }

    "MovingX apply" should "return 2 for moving right" in {
        (envStub.getMarioMovement _) when() returns(Array(1, -1))

        assertResult(2: Byte) {
            MovingX(envStub)
        }
        assert(MovingX.RIGHT == (2: Byte))
    }

    "MovingY apply" should "return 0 for not moving" in {
        (envStub.getMarioMovement _) when() returns(Array(1, 0))

        assertResult(0: Byte) {
            MovingY(envStub)
        }
        assert(MovingY.STILL == (0: Byte))
    }

    "MovingY apply" should "return 1 for moving down" in {
        (envStub.getMarioMovement _) when() returns(Array(-1, 1))

        assertResult(1: Byte) {
            MovingY(envStub)
        }
        assert(MovingY.DOWN == (1: Byte))
    }

    "MovingY apply" should "return 2 for moving up" in {
        (envStub.getMarioMovement _) when() returns(Array(0, -1))

        assertResult(2: Byte) {

```

```

        MovingY(envStub)
    }
    assert(MovingY.UP == (2: Byte))
}

"EnemyLeft apply" should "return 1 if enemy in range to the left of mario" in {
    val enemy = RandomUtils.randomIntTupleInclusive(EnemyLeft.AREA_UL, EnemyLeft.AREA_BR)
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(1: Byte) {
        EnemyLeft(envStub)
    }
}

"EnemyLeft apply" should "return 0 if enemy too far to the left of mario" in {
    val enemy = (EnemyLeft.AREA_UL._1 + 1, EnemyLeft.AREA_UL._2 - 1) //One Down, One Left
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        EnemyLeft(envStub)
    }
}

"EnemyLeft apply" should "return 0 if enemy is to right of mario" in {
    val enemy = (0, 2)
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        EnemyLeft(envStub)
    }
}

"EnemyUpperRight apply" should "return 1 if enemy in range to the top right of mario part 1" in {
    val enemy = RandomUtils.randomIntTupleInclusive(EnemyUpperRight.AREA_1_UL, EnemyUpperRight.AREA_1_BR)
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(1: Byte) {
        EnemyUpperRight(envStub)
    }
}

"EnemyUpperRight apply" should "return 1 if enemy in range to the top right of mario part 2" in {
    val enemy = RandomUtils.randomIntTupleInclusive(EnemyUpperRight.AREA_2_UL, EnemyUpperRight.AREA_2_BR)
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(1: Byte) {
        EnemyUpperRight(envStub)
    }
}

"EnemyUpperRight apply" should "return 0 if enemy is lower right of Mario" in {
    val enemy = (1, 1)
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        EnemyUpperRight(envStub)
    }
}

"EnemyUpperRight apply" should "return 0 if enemy is to the left mario" in {
    val enemy = (0, -1)
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        EnemyUpperRight(envStub)
    }
}

"EnemyLowerRight apply" should "return 1 if enemy in range to the top right of mario part 1" in {
    val enemy = RandomUtils.randomIntTupleInclusive(EnemyLowerRight.AREA_1_UL, EnemyLowerRight.AREA_1_BR)
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(1: Byte) {
        EnemyLowerRight(envStub)
    }
}

```

```

"EnemyLowerRight apply" should "return 1 if enemy in range to the top right of mario part 2" in {
    val enemy = RandomUtils.randomIntTupleInclusive(EnemyLowerRight.AREA_2_UL, EnemyLowerRight.AREA_2_BR)
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(1: Byte) {
        EnemyLowerRight(envStub)
    }
}

"EnemyLowerRight apply" should "return 0 if enemy is upper right of Mario" in {
    val enemy = (-1, -1)
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        EnemyLowerRight(envStub)
    }
}

"EnemyLowerRight apply" should "return 0 if enemy is to the left mario" in {
    val enemy = (0, -1)
    (envStub.getEnemiesObservationZ _) when(*) returns(EnvironmentFixtures.getEnemyScene(enemy))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        EnemyLowerRight(envStub)
    }
}

"PitAhead apply" should "return 1 if there is a pit in close range ahead of mario" in {
    val pitCols = Set(1, 2, 3);
    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getLevelSceneWithPit(1, pitCols))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(1: Byte) {
        PitAhead(envStub)
    }
}

"PitAhead apply" should "return 2 if there is a pit in range but not close ahead of mario" in {
    val pitCols = Set(2, 3, 4);
    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getLevelSceneWithPit(1, pitCols))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(2: Byte) {
        PitAhead(envStub)
    }
}

"PitAhead apply" should "return 1 if mario below terrain" in {
    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.baseLevelScene(-2))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(1: Byte) {
        PitAhead(envStub)
    }
}

"PitAhead apply" should "return 0 if there are no pits on screen" in {
    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.baseLevelScene(1))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        PitAhead(envStub)
    }
}

"PitAhead apply" should "return 0 if there are only pits behind mario" in {
    val pitCols = Set(-7, -6, -3, -2, -1)
    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getLevelSceneWithPit(3, pitCols))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        PitAhead(envStub)
    }
}

"PitBelow apply" should "return 1 if there is a pit below mario" in {
    val pitCols = Set(0, 1)
    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getLevelSceneWithPit(3, pitCols))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

```

```

    assertResult(1: Byte) {
        PitBelow(envStub)
    }
}

"PitBelow apply" should "return 0 if there is a pit too far ahead of mario" in {
    val pitCols = Set(PitBelow.COL_R+2, PitBelow.COL_R+3, PitBelow.COL_R+4)
    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getLevelSceneWithPit(1, pitCols))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        PitBelow(envStub)
    }
}

"PitBelow apply" should "return 0 if there is a pit too far behind of mario" in {
    val pitCols = Set(PitBelow.COL_L-2, PitBelow.COL_L-3, PitBelow.COL_L-4)
    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getLevelSceneWithPit(1, pitCols))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        PitBelow(envStub)
    }
}

"ObstacleAhead apply" should "return 1 if there is terrain in range to right of mario" in {
    val groundLevels = Set(1, 2);
    val additions = Set((0, RandomUtils.randomIntIn(ObstacleAhead.AREA_OBS_UL._2, ObstacleAhead.AREA_OBS_BR._2)))

    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getCustomLevelScene(groundLevels,
        Set.empty, additions))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(1: Byte) {
        ObstacleAhead(envStub)
    }
}

"ObstacleAhead apply" should "return 0 if there is terrain to left of mario" in {
    val groundLevels = Set(1, 2);
    val additions = Set((0, -1))

    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getCustomLevelScene(groundLevels,
        Set.empty, additions))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        ObstacleAhead(envStub)
    }
}

"ObstacleAhead apply" should "return 0 if only terrain obstacles are too high above mario" in {
    val groundLevels = Set(1, 2);
    val additions = Set((ObstacleAhead.AREA_OBS_UL._1-1, RandomUtils.randomIntIn(ObstacleAhead.AREA_OBS_UL._2,
        ObstacleAhead.AREA_OBS_BR._2)))

    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getCustomLevelScene(groundLevels,
        Set.empty, additions))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        ObstacleAhead(envStub)
    }
}

"ObstacleAhead apply" should "return 0 if only terrain falls away" in {
    val groundLevels = Set(2,3);
    val additions = Set((1,1), (1,0), (1,-1)) //Mario on small platform

    (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getCustomLevelScene(groundLevels,
        Set.empty, additions))
    (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

    assertResult(0: Byte) {
        ObstacleAhead(envStub)
    }
}

"ObstacleAhead apply" should "return 1 if there is terrain falls and rises to right of mario (is stepped)" in {
    val groundLevels = Set(1, 2);
    val randomIntInColRange = RandomUtils.randomIntEx(ObstacleAhead.AREA_OBS_UL._2, ObstacleAhead.AREA_OBS_BR._2)
    val removals = Set((1, randomIntInColRange), (1, randomIntInColRange+1)) //Small dip in front of mario

```

```

        (envStub.getLevelSceneObservationZ _) when(*) returns(EnvironmentFixtures.getCustomLevelScene(groundLevels,
            removals, Set.empty))
        (envStub.getMarioEgoPos _) when() returns(Array(EnvironmentFixtures.mario._1, EnvironmentFixtures.mario._2))

        assertResult(1: Byte) {
            ObstacleAhead(envStub)
        }
    }
}

```

---

```

package com.montywest.marioai.rules

import org.scalatest.BeforeAndAfterAll
import org.scalamock.scalatest.MockFactory
import org.scalatest.Matchers
import org.scalatest.FlatSpec
import ch.idsia.benchmark.mario.environments.Environment
import ch.idsia.benchmark.mario.engine.sprites.Mario
import com.montywest.marioai.fixtures.RulePackageFixtures

class RuleTest extends FlatSpec with Matchers with MockFactory with BeforeAndAfterAll {

    "getCondition" should "return Conditions of length equal to number of perceptions" in {
        assert(Perception.NUMBER_OF_PERCEPTIONS == Rule.BLANK_RULE.getConditions.length)
    }

    "getExAction" should "return ExAction of length equal to number of keys" in {
        assert(Environment.numberofKeys == Rule.BLANK_RULE.getExAction.length)
    }

    "getMWAction" should "return MWAction of length equal to MWAction length" in {
        assert(MWAction.LENGTH == Rule.BLANK_RULE.getMWAction.length)
    }

    "getVectorRep" should "return Vector of bytes that represents both conditions and action" in {
        val vec = Rule.build(Map(MarioMode -> MarioMode.FIRE,
            EnemyLowerRight -> EnemyLowerRight.FALSE,
            MovingY -> MovingY.DOWN),
            Set(KeyLeft, KeyJump)).getVectorRep

        assert(MWAction.LENGTH + Conditions.LENGTH == vec.length)
        assert(vec == Vector(2, -1, -1, -1, -1, 0, -1, -1, -1, -1, 1, 0, 1, 0, 0))
    }

    "build" should "take map of conditions and create rule the gives the correct Conditions instance" in {
        val conditions = Rule.build(Map(MarioMode -> MarioMode.FIRE,
            EnemyLowerRight -> EnemyLowerRight.FALSE,
            MovingY -> MovingY.DOWN),
            Set.empty).getConditions
        assert(MarioMode.FIRE == conditions(MarioMode.index))
        assert(EnemyLowerRight.FALSE == conditions(EnemyLowerRight.index))
        assert(MovingY.DOWN == conditions(MovingY.index))

        assert(conditions.zipWithIndex.forall { case (b: Byte, i: Int) => {
            Set(MarioMode.index, EnemyLowerRight.index, MovingY.index).contains(i) ||
            b == Conditions.DONT_CARE
        }})
    }

    "build" should "validate condition map to test bytes are in perception range" in {
        intercept[Exception] {
            val conditions = Rule.build(Map(MarioMode -> MarioMode.FIRE,
            EnemyLowerRight -> EnemyLowerRight.FALSE,
            MovingY -> (MovingY.limit + 5).toByte),
            Set.empty).getConditions
        }
    }

    "build" should "take set of actions and create rule that gives correct the ExActions instance" in {
        val action = Rule.build(Map.empty,
            Set(KeyLeft, KeyJump)).getExAction

        assert(action(Mario.KEY_LEFT))
        assert(action(Mario.KEY_JUMP))
        assert(!action(Mario.KEY_DOWN))
        assert(!action(Mario.KEY_RIGHT))
        assert(!action(Mario.KEY_UP))
        assert(!action(Mario.KEY_SPEED))
    }
}

```



```

    "build" should "take set of actions and create rule that gives correct the MWActions instance" in {
        val action = Rule.build(Map.empty,
            Set(KeyLeft, KeyJump)).getMWAction

        assert(action(MWAction.LEFT_INDEX) == MWAction.ACTION_TRUE)
        assert(action(MWAction.JUMP_INDEX) == MWAction.ACTION_TRUE)

        assert(action(MWAction.RIGHT_INDEX) == MWAction.ACTION_FALSE)
        assert(action(MWAction.SPEED_INDEX) == MWAction.ACTION_FALSE)
    }

    "score" should "return count observations that match conditions" in {
        val rule = Rule.build(Map(MarioMode -> MarioMode.FIRE,
            EnemyLowerRight -> EnemyLowerRight.FALSE,
            MovingY -> MovingY.DOWN),
            Set.empty)

        val observation = RulePackageFixtures.getObservation(
            Map(MarioMode -> MarioMode.FIRE,
                EnemyLowerRight -> EnemyLowerRight.FALSE,
                MovingY -> MovingY.DOWN,
                MovingX -> MovingX.RIGHT,
                PitBelow -> PitBelow.TRUE)
        )

        assertResult(3: Int) {
            rule.scoreAgainst(observation)
        }
    }

    "score" should "return -1 if any unmatching conditions" in {
        val rule = Rule.build(Map(MarioMode -> MarioMode.FIRE,
            EnemyLowerRight -> EnemyLowerRight.FALSE,
            MovingY -> MovingY.DOWN),
            Set.empty)

        val observation = RulePackageFixtures.getObservation(
            Map(MarioMode -> MarioMode.FIRE,
                EnemyLowerRight -> EnemyLowerRight.FALSE,
                MovingY -> MovingY.UP,
                MovingX -> MovingX.RIGHT,
                PitBelow -> PitBelow.TRUE)
        )

        assertResult(-1: Int) {
            rule.scoreAgainst(observation)
        }
    }
}

```

---

```

package com.montywest.marioai.rules

import org.scalatest.BeforeAndAfterAll
import org.scalatest.mock.MockFactory
import org.scalatest.Matchers
import org.scalatest.FlatSpec

class RulesetTest extends FlatSpec with Matchers with MockFactory with BeforeAndAfterAll {

    "apply" should "take (a sequence of rules and) a default action as be able construct ruleset with correct MWAction" in {
        val defaultAction: MWAction = Ruleset(Seq(Rule.BLANK_RULE), Set(KeyLeft, KeyJump)).defaultAction

        val expected: Vector[Byte] = Vector.tabulate(4)(_ match {
            case MWAction.LEFT_INDEX => MWAction.ACTION_TRUE
            case MWAction.JUMP_INDEX => MWAction.ACTION_TRUE
            case _ => MWAction.ACTION_FALSE
        })

        assert(expected == defaultAction);
    }

    "build" should "take a vector of bytes and produce ruleset (with default action)" in {
        val vec = Vector[Byte](-1, 1, 1, 1, 1, 0, -1, -1, -1, 1, 1, 0, 2, 1, 0, 1, 0,
            1, -1, 0, 0, 1, 1, -1, 1, -1, -1, 1, 0, 0, 1, 0,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 1, 0, 0)

        val ruleset = Ruleset.build(vec)
    }
}

```

```

    assert(Rule(Vector[Byte](-1, 1, 1, 1, 0, -1, -1, 1, 0, 2), Vector(1, 0, 1, 0)) == ruleset.rules(0))
    assert(Rule(Vector[Byte](1, -1, 0, 0, 1, 1, -1, 1, -1, -1), Vector(0, 0, 1, 0)) == ruleset.rules(1))
    assert(Vector(0, 1, 0, 0) == ruleset.defaultAction)
}

"build" should "take a vector of bytes and produce ruleset (with specified default action)" in {
    val vec = Vector[Byte](-1, 1, 1, 1, 0, -1, -1, 1, 0, 2, 1, 0, 1, 0,
        1, -1, 0, 0, 1, 1, -1, 1, -1, -1, 0, 0, 1, 0,
        2, -1, -1, 1, 1, -1, -1, 0, -1, -1, 0, 1, 0, 0)

    val ruleset = Ruleset.build(vec, Vector(0, 1, 1, 0))

    assert(Rule(Vector[Byte](-1, 1, 1, 1, 0, -1, -1, 1, 0, 2), Vector(1, 0, 1, 0)) == ruleset.rules(0))
    assert(Rule(Vector[Byte](1, -1, 0, 0, 1, 1, -1, 1, -1, -1), Vector(0, 0, 1, 0)) == ruleset.rules(1))
    assert(Rule(Vector[Byte](2, -1, -1, 1, 1, -1, -1, 0, -1, -1), Vector(0, 1, 0, 0)) == ruleset.rules(2))
    assert(Vector(0, 1, 1, 0) == ruleset.defaultAction)
}

"build" should "take a vector of bytes and produce ruleset (without any default action)" in {
    val vec = Vector[Byte](-1, 1, 1, 1, 0, -1, -1, 1, 0, 2, 1, 0, 1, 0,
        1, -1, 0, 0, 1, 1, -1, 1, -1, -1, 0, 0, 1, 0,
        2, -1, -1, 1, 1, -1, -1, 0, -1, -1, 0, 1, 0, 0)

    val ruleset = Ruleset.build(vec)

    assert(Rule(Vector[Byte](-1, 1, 1, 1, 0, -1, -1, 1, 0, 2), Vector(1, 0, 1, 0)) == ruleset.rules(0))
    assert(Rule(Vector[Byte](1, -1, 0, 0, 1, 1, -1, 1, -1, -1), Vector(0, 0, 1, 0)) == ruleset.rules(1))
    assert(Rule(Vector[Byte](2, -1, -1, 1, 1, -1, -1, 0, -1, -1), Vector(0, 1, 0, 0)) == ruleset.rules(2))
    assert(Ruleset.FALLBACK_ACTION == ruleset.defaultAction)
}

"build (with default) then getVectorRep (with default)" should "be opposite operations" in {
    val vec = Vector[Byte](-1, 1, 1, 1, 0, -1, -1, 1, 0, 2, 1, 0, 1, 0,
        1, -1, 0, 0, 1, 1, -1, 1, -1, -1, 0, 0, 1, 0,
        -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 1, 0, 0)

    val ruleset = Ruleset.build(vec)

    assert(vec == ruleset.getVectorRep(true));
}

"build (without default) then getVectorRep (without default)" should "be opposite operations" in {
    val vec = Vector[Byte](-1, 1, 1, 1, 0, -1, -1, 1, 0, 2, 1, 0, 1, 0,
        1, -1, 0, 0, 1, 1, -1, 1, -1, -1, 0, 0, 1, 0,
        2, -1, -1, 1, 1, -1, -1, 0, -1, -1, 0, 1, 0, 0)

    val ruleset = Ruleset.build(vec)

    assert(vec == ruleset.getVectorRep(false));
}

"getBestExAction" should "return best fitting action as ExAction" in {
    val vec = Vector[Byte](0, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0, 1,
        0, 0, -1, 0, -1, -1, -1, -1, -1, -1, 0, 0, 1, 0,
        0, 0, 0, -1, -1, -1, -1, -1, -1, -1, 0, 0, 1, 1,
        0, 0, 0, -1, 0, 0, -1, -1, -1, -1, 0, 1, 0, 0,
        0, 0, 0, -1, 0, 0, 1, -1, -1, 0, 1, 0, 1,
        0, 0, -1, -1, -1, -1, -1, -1, -1, 0, 1, 1, 1,
        -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 0, 0, 0)

    val ruleset = Ruleset.build(vec)

    assert(ExAction(Vector[Byte](0, 1, 0, 0)).deep == ruleset.getBestExAction(Vector[Byte](0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0)).deep)
    assert(ExAction(Vector[Byte](0, 0, 0, 1)).deep == ruleset.getBestExAction(Vector[Byte](0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 0)).deep)
    assert(ExAction(Vector[Byte](0, 0, 1, 0)).deep == ruleset.getBestExAction(Vector[Byte](0, 0, 0, 0, 1, 1, 1, 1,
        1, 1, 1)).deep)
    assert(ExAction(Vector[Byte](0, 1, 1, 1)).deep == ruleset.getBestExAction(Vector[Byte](0, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1)).deep)
    assert(ExAction(Vector[Byte](1, 0, 0, 0)).deep == ruleset.getBestExAction(Vector[Byte](1, 0, 1, 0, 1, 0, 1, 0,
        1, 0, 1)).deep)
}

```

```
}
```

## F.2 Mario AI Benchmark Modifications

```
/*
 * Copyright (c) 2009-2010, Sergey Karakovskiy and Julian Togelius
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 * * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 * * Neither the name of the Mario AI nor the
 * names of its contributors may be used to endorse or promote products
 * derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 * IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package ch.idsia.benchmark.mario.environments;

import ch.idsia.agents.Agent;
import ch.idsia.benchmark.mario.engine.*;
import ch.idsia.benchmark.mario.engine.level.Level;
import ch.idsia.benchmark.mario.engine.sprites.Mario;
import ch.idsia.benchmark.mario.engine.sprites.Sprite;
import ch.idsia.benchmark.tasks.SystemOfValues;
import ch.idsia.tools.EvaluationInfo;
import ch.idsia.tools.MarioAIOptions;
import ch.idsia.tools.punj.PunctualJudge;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by IntelliJ IDEA.
 * User: Sergey Karakovskiy, sergey@idsia.ch
 * Date: Mar 3, 2010 Time: 10:08:13 PM
 * Package: ch.idsia.benchmark.mario.environments
 */

public final class MarioEnvironment implements Environment
{
    private int[] marioEgoPos = new int[2];
    private int[] marioMovement = new int[2];

    private int receptiveFieldHeight = -1; // to be setup via MarioAIOptions
    private int receptiveFieldWidth = -1; // to be setup via MarioAIOptions
    private int prevRFH = -1;
    private int prevRFW = -1;

    private byte[][] levelSceneZ; // memory is allocated in reset
    private byte[][] enemiesZ; // memory is allocated in reset
    private byte[][] mergedZZ; // memory is allocated in reset

    public List<Sprite> sprites;

    private int[] serializedLevelScene; // memory is allocated in reset
    private int[] serializedEnemies; // memory is allocated in reset
    private int[] serializedMergedObservation; // memory is allocated in reset

    private final LevelScene levelScene;
    // private int frame = 0;
    private MarioVisualComponent marioVisualComponent;
    private Agent agent;
```

```

private static final MarioEnvironment ourInstance = new MarioEnvironment();
private static final EvaluationInfo evaluationInfo = new EvaluationInfo();

private static String marioTraceFile;

private Recorder recorder;

public static SystemOfValues IntermediateRewardsSystemOfValues = new SystemOfValues();

DecimalFormat df = new DecimalFormat("#####.##");

public static MarioEnvironment getInstance()
{
    return ourInstance;
}

private MarioEnvironment()
{
    // System.out.println("System.getProperty(\"java.awt.headless\") = " + System.getProperty("java.awt.headless"));
    // System.out.println("System.getProperty(\"verbose\") = " + System.getProperty("-verbose"));
    // System.out.println("Java: JA ZDES'!!");
    // System.out.flush();
    // System.out.println(GlobalOptions.getBenchmarkName());
    levelScene = new LevelScene();
}

public void resetDefault()
{
    levelScene.resetDefault();
}

public void reset(String args)
{
    MarioAIOptions marioAIOptions = MarioAIOptions.getOptionsByString(args);
    this.reset(marioAIOptions);
    // MarioAIOptions opts = new MarioAIOptions(setUpOptions);
    // int[] intOpts = opts.toIntArray();
    // this.reset(intOpts);
}

public void reset(MarioAIOptions setUpOptions)
{
    /*System.out.println("\nsetUpOptions = " + setUpOptions);
    for (int i = 0; i < setUpOptions.length; ++i)
    {
        System.out.print(" op[" + i + "] = " + setUpOptions[i]);
    }
    System.out.println("");
    System.out.flush();*/
    // if (!setUpOptions.getReplayOptions().equals(""))

    this.setAgent(setUpOptions.getAgent());

    receptiveFieldWidth = setUpOptions.getReceptiveFieldWidth();
    receptiveFieldHeight = setUpOptions.getReceptiveFieldHeight();

    if (receptiveFieldHeight != this.prevRFH || receptiveFieldWidth != this.prevRFW)
    {
        serializedLevelScene = new int[receptiveFieldHeight * receptiveFieldWidth];
        serializedEnemies = new int[receptiveFieldHeight * receptiveFieldWidth];
        serializedMergedObservation = new int[receptiveFieldHeight * receptiveFieldWidth];

        levelSceneZ = new byte[receptiveFieldHeight][receptiveFieldWidth];
        enemiesZ = new byte[receptiveFieldHeight][receptiveFieldWidth];
        mergedZZ = new byte[receptiveFieldHeight][receptiveFieldWidth];
        this.prevRFH = this.receptiveFieldHeight;
        this.prevRFW = this.receptiveFieldWidth;
    }

    marioEgoPos[0] = setUpOptions.getMarioEgoPosRow();
    marioEgoPos[1] = setUpOptions.getMarioEgoPosCol();

    if (marioEgoPos[0] == 9 && getReceptiveFieldWidth() != 19)
        marioEgoPos[0] = getReceptiveFieldWidth() / 2;
    if (marioEgoPos[1] == 9 && getReceptiveFieldHeight() != 19)
        marioEgoPos[1] = getReceptiveFieldHeight() / 2;

    marioTraceFile = setUpOptions.getTraceFileName();

    if (setUpOptions.isVisualization())
    {
        if (marioVisualComponent == null)
            marioVisualComponent = MarioVisualComponent.getInstance(setUpOptions, this);
        levelScene.reset(setUpOptions);
    }
}

```

```

        marioVisualComponent.reset();
        marioVisualComponent.postInitGraphicsAndLevel();
        marioVisualComponent.setAgent(agent);
        marioVisualComponent.setLocation(setUpOptions.getViewLocation());
        marioVisualComponent.setAlwaysOnTop(setUpOptions.isViewAlwaysOnTop());

        if (setUpOptions.isScale2X())
            GlobalOptions.changeScale2x();
    } else
        levelScene.reset(setUpOptions);

    sprites = levelScene.sprites;

    //create recorder
    String recordingFileName = setUpOptions.getRecordingFileName();

    if (!recordingFileName.equals("off"))
    {
        if (recordingFileName.equals("on"))
            recordingFileName = GlobalOptions.getTimeStamp() + ".zip";

        try
        {
            if (recordingFileName.equals("lazy"))
                recorder = new Recorder();
            else
                recorder = new Recorder(recordingFileName);

            recorder.createFile("level.lvl");
            recorder.writeObject(levelScene.level);
            recorder.closeFile();

            recorder.createFile("options");
            recorder.writeObject(setUpOptions.asString());
            recorder.closeFile();

            recorder.createFile("actions.act");
        } catch (FileNotFoundException e)
        {
            System.err.println("[Mario AI EXCEPTION] : Some of the recording components were not created.
                                Recording failed");
        } catch (IOException e)
        {
            System.err.println("[Mario AI EXCEPTION] : Some of the recording components were not created.
                                Recording failed");
            e.printStackTrace();
        }
    }
    evaluationInfo.reset();
    PunctualJudge.resetCounter();
}

public void tick()
{
    levelScene.tick();
    if (GlobalOptions.isVisualization)
        marioVisualComponent.tick();
}

public float[] getMarioFloatPos()
{
    return levelScene.getMarioFloatPos();
}

public int getMarioMode()
{
    return levelScene.getMarioMode();
}

public byte[][] getLevelSceneObservationZ(int ZLevel)
{
    int mCol = marioEgoPos[1];
    int mRow = marioEgoPos[0];
    for (int y = levelScene.mario.mapY - mRow, row = 0; y <= levelScene.mario.mapY + (receptiveFieldHeight - mRow - 1); y++, row++)
    {
        for (int x = levelScene.mario.mapX - mCol, col = 0; x <= levelScene.mario.mapX + (receptiveFieldWidth - mCol - 1); x++, col++)
        {
            if (x >= 0 && x < levelScene.level.length && y >= 0 && y < levelScene.level.height)
            {
                mergedZZ[row][col] = levelSceneZ[row][col] = GeneralizerLevelScene.ZLevelGeneralization(levelScene
                    .level.map[x][y], ZLevel);
            } else
            {

```

```

        {
            mergedZZ[row][col] = levelSceneZ[row][col] = 0;
        }
    }
}
return levelSceneZ;
}

public byte[][] getEnemiesObservationZ(int ZLevel)
{
    int marioEgoCol = marioEgoPos[1];
    int marioEgoRow = marioEgoPos[0];
    for (int w = 0; w < enemiesZ.length; w++)
        for (int h = 0; h < enemiesZ[0].length; h++)
            enemiesZ[w][h] = 0;
    for (Sprite sprite : sprites)
    {
        if (sprite.isDead() || sprite.kind == levelScene.mario.kind)
            continue;
        if (sprite.mapX >= 0 &&
            sprite.mapX >= levelScene.mario.mapX - marioEgoCol &&
            sprite.mapX <= levelScene.mario.mapX + (receptiveFieldWidth - marioEgoCol - 1) &&
            sprite.mapY >= 0 &&
            sprite.mapY >= levelScene.mario.mapY - marioEgoRow &&
            sprite.mapY <= levelScene.mario.mapY + (receptiveFieldHeight - marioEgoRow - 1) &&
            sprite.kind != Sprite.KIND_PRINCESS)
        {
            int row = sprite.mapY - levelScene.mario.mapY + marioEgoRow;
            int col = sprite.mapX - levelScene.mario.mapX + marioEgoCol;
            // TODO: !H! take care about side effects of line 243 and be sure not to contaminate
            // levelSceneObservation
            mergedZZ[row][col] = enemiesZ[row][col] = GeneralizerEnemies.ZLevelGeneralization(sprite.kind, ZLevel);
        }
    }
    return enemiesZ;
}
// TODO: !H! substitute the content of getMergedObservationZZ by getLevelSceneObservationZ,
// TODO: !H! getEnemiesObservationZ, called one after another!

public byte[][] getMergedObservationZZ(int ZLevelScene, int ZLevelEnemies)
{
    // int MarioXInMap = (int) mario.x / cellSize;
    // int MarioYInMap = (int) mario.y / cellSize;

    // if (MarioXInMap != (int) mario.x / cellSize || MarioYInMap != (int) mario.y / cellSize )
    // throw new Error("WRONG mario x or y pos");
    int mCol = marioEgoPos[1];
    int mRow = marioEgoPos[0];
    for (int y = levelScene.mario.mapY - mRow/*receptiveFieldHeight / 2*/, row = 0; y <= levelScene.mario.mapY +
        (receptiveFieldHeight - mRow - 1)/*receptiveFieldHeight / 2*/; y++, row++)
    {
        for (int x = levelScene.mario.mapX - mCol/*receptiveFieldWidth / 2*/, col = 0; x <= levelScene.mario.mapX
            + (receptiveFieldWidth - mCol - 1)/*receptiveFieldWidth / 2*/; x++, col++)
        {
            if (x >= 0 && x < levelScene.level.xExit && y >= 0 && y < levelScene.level.height)
            {
                mergedZZ[row][col] = GeneralizerLevelScene.ZLevelGeneralization(levelScene.level.map[x][y],
                    ZLevelScene);
            }
            else
                mergedZZ[row][col] = 0;
        }
        // if (x == MarioXInMap && y == MarioYInMap)
        // mergedZZ[row][col] = mario.kind;
    }
    // for (int w = 0; w < mergedZZ.length; w++)
    // for (int h = 0; h < mergedZZ[0].length; h++)
    // mergedZZ[w][h] = -1;
    for (Sprite sprite : sprites)
    {
        if (sprite.isDead() || sprite.kind == levelScene.mario.kind)
            continue;
        if (sprite.mapX >= 0 &&
            sprite.mapX >= levelScene.mario.mapX - mCol &&
            sprite.mapX <= levelScene.mario.mapX + (receptiveFieldWidth - mCol - 1) &&
            sprite.mapY >= 0 &&
            sprite.mapY >= levelScene.mario.mapY - mRow &&
            sprite.mapY <= levelScene.mario.mapY + (receptiveFieldHeight - mRow - 1) &&
            sprite.kind != Sprite.KIND_PRINCESS)
        {
            int row = sprite.mapY - levelScene.mario.mapY + mRow;
            int col = sprite.mapX - levelScene.mario.mapX + mCol;
            byte tmp = GeneralizerEnemies.ZLevelGeneralization(sprite.kind, ZLevelEnemies);

```

```

        if (tmp != Sprite.KIND_NONE)
            mergedZZ[row][col] = tmp;
    }
}

return mergedZZ;
}

public List<String> getObservationStrings(boolean Enemies, boolean LevelMap,
    boolean mergedObservationFlag,
    int ZLevelScene, int ZLevelEnemies)
{
    List<String> ret = new ArrayList<String>();
    if (levelScene.level != null && levelScene.mario != null)
    {
        ret.add("Total levelScene length = " + levelScene.level.length);
        ret.add("Total levelScene height = " + levelScene.level.height);
        ret.add("Physical Mario Position (x,y): (" + df.format(levelScene.mario.x) + "," + df.format(levelScene.
            mario.y) + ")");
        ret.add("Mario Observation (Receptive Field) Width: " + receptiveFieldWidth + " Height: " +
            receptiveFieldHeight);
        ret.add("X Exit Position: " + levelScene.level.xExit);
        int MarioXInMap = (int) levelScene.mario.x / levelScene.cellSize; //TODO: !!H! doublecheck and replace
            with levelScene.mario.mapX
        int MarioYInMap = (int) levelScene.mario.y / levelScene.cellSize; //TODO: !!H! doublecheck and replace
            with levelScene.mario.mapY
        ret.add("Calibrated Mario Position (x,y): (" + MarioXInMap + "," + MarioYInMap + ")\n");

        byte[] [] levelScene = getLevelSceneObservationZ(ZLevelScene);
        if (LevelMap)
        {
            ret.add("~ZLevel: Z" + ZLevelScene + " map:\n");
            for (int x = 0; x < levelScene.length; ++x)
            {
                String tmpData = "";
                for (int y = 0; y < levelScene[0].length; ++y)
                    tmpData += levelSceneCellToString(levelScene[x][y]);
                ret.add(tmpData);
            }
        }

        byte[] [] enemiesObservation = null;
        if (Enemies || mergedObservationFlag)
            enemiesObservation = getEnemiesObservationZ(ZLevelEnemies);

        if (Enemies)
        {
            ret.add("~ZLevel: Z" + ZLevelScene + " Enemies Observation:\n");
            for (int x = 0; x < enemiesObservation.length; x++)
            {
                String tmpData = "";
                for (int y = 0; y < enemiesObservation[0].length; y++)
                {
                    // if (x >= 0 && x <= level.xExit)
                    tmpData += enemyToStr(enemiesObservation[x][y]);
                }
                ret.add(tmpData);
            }
        }

        if (mergedObservationFlag)
        {
            byte[] [] mergedObs = getMergedObservationZZ(ZLevelScene, ZLevelEnemies);
            ret.add("~ZLevelScene: Z" + ZLevelScene + " ZLevelEnemies: Z" + ZLevelEnemies + " ; Merged observation
                /* Mario ^> #M.# */");
            for (int x = 0; x < levelScene.length; ++x)
            {
                String tmpData = "";
                for (int y = 0; y < levelScene[0].length; ++y)
                    tmpData += levelSceneCellToString(mergedObs[x][y]);
                ret.add(tmpData);
            }
        }
    }
    else
        ret.add("~[MarioAI ERROR] level : " + levelScene.level + " mario : " + levelScene.mario);
    return ret;
}

private String levelSceneCellToString(int el)
{
    String s = "";
    if (el == 0 || el == 1)
        s = "##";
}

```

```

        s += (el == levelScene.mario.kind) ? "#M.#" : el;
        while (s.length() < 4)
            s += "#";

        return s + " ";
    }

    private String enemyToStr(int el)
    {
        String s = "";
        if (el == 0)
            s = "";
        s += (el == levelScene.mario.kind) ? "-m" : el;
        while (s.length() < 2)
            s += "#";
        return s + " ";
    }

    public float[] getEnemiesFloatPos()
    {
        return levelScene.getEnemiesFloatPos();
    }

    public boolean isMarioOnGround()
    {
        return levelScene.isMarioOnGround();
    }

    public boolean isMarioAbleToJump()
    {
        return levelScene.isMarioAbleToJump();
    }

    public boolean isMarioCarrying()
    {
        return levelScene.isMarioCarrying();
    }

    public boolean isMarioAbleToShoot()
    {
        return levelScene.isMarioAbleToShoot();
    }

    public int getReceptiveFieldWidth()
    {
        return receptiveFieldWidth;
    }

    public int getReceptiveFieldHeight()
    {
        return receptiveFieldHeight;
    }

    public int getKillsTotal()
    {
        return levelScene.getKillsTotal();
    }

    public int getKillsByFire()
    {
        return levelScene.getKillsByFire();
    }

    public int getKillsByStomp()
    {
        return levelScene.getKillsByStomp();
    }

    public int getKillsByShell()
    {
        return levelScene.getKillsByShell();
    }

    public int getMarioStatus()
    {
        return levelScene.getMarioStatus();
    }

    public int[] getObservationDetails()
    {
        return new int[]{receptiveFieldWidth, receptiveFieldHeight, marioEgoPos[0], marioEgoPos[1]};
    }

```



```

public List<Sprite> getSprites()
{
    return sprites;
}

public int[] getSerializedFullObservationZZ(int ZLevelScene, int ZLevelEnemies)
{
    int[] obs = new int[receptiveFieldHeight * receptiveFieldWidth * 2 + 11]; // 11 is a size of the MarioState array

    int receptiveFieldSize = receptiveFieldWidth * receptiveFieldHeight;

    System.arraycopy(getSerializedLevelSceneObservationZ(ZLevelScene), 0, obs, 0, receptiveFieldSize);
    System.arraycopy(getSerializedEnemiesObservationZ(ZLevelScene), 0, obs, receptiveFieldSize,
        receptiveFieldSize);
    System.arraycopy(getMarioState(), 0, obs, receptiveFieldSize * 2, 11);

    return obs;
}

public int[] getSerializedLevelSceneObservationZ(int ZLevelScene)
{
    // serialization into arrays of primitive types to speed up the data transfer.
    byte[] levelScene = this.getLevelSceneObservationZ(ZLevelScene);
    for (int i = 0; i < serializedLevelScene.length; ++i)
    {
        final int i1 = i / receptiveFieldWidth;
        final int i2 = i % receptiveFieldWidth;
        serializedLevelScene[i] = (int) levelScene[i1][i2];
    }
    return serializedLevelScene;
}

public int[] getSerializedEnemiesObservationZ(int ZLevelEnemies)
{
    // serialization into arrays of primitive types to speed up the data transfer.
    byte[][] enemies = this.getEnemiesObservationZ(ZLevelEnemies);
    for (int i = 0; i < serializedEnemies.length; ++i)
        serializedEnemies[i] = (int) enemies[i / receptiveFieldWidth][i % receptiveFieldWidth];
    return serializedEnemies;
}

public int[] getSerializedMergedObservationZZ(int ZLevelScene, int ZLevelEnemies)
{
    // serialization into arrays of primitive types to speed up the data transfer.
    byte[][] merged = this.getMergedObservationZZ(ZLevelScene, ZLevelEnemies);
    for (int i = 0; i < serializedMergedObservation.length; ++i)
        serializedMergedObservation[i] = (int) merged[i / receptiveFieldWidth][i % receptiveFieldWidth];
    return serializedMergedObservation;
}

public float[] getCreaturesFloatPos()
{
    return levelScene.getCreaturesFloatPos();
}

public int[] getMarioState()
{
    return levelScene.getMarioState();
}

public void performAction(boolean[] action)
{
    try
    {
        if (recorder != null && recorder.canRecord() && action != null)
        {
            recorder.writeAction(action);
            recorder.changeRecordingState(GlobalOptions.isRecording, getTimeSpent());
        }
    } catch (IOException e)
    {
        e.printStackTrace();
    }
    levelScene.performAction(action);
}

public boolean isLevelFinished()
{
    return levelScene.isLevelFinished();
}

public int[] getEvaluationInfoAsInts()
{

```

```

        return this.getEvaluationInfo().toIntArray();
    }

    public String getEvaluationInfoAsString()
    {
        return this.getEvaluationInfo().toString();
    }

    public EvaluationInfo getEvaluationInfo()
    {
        computeEvaluationInfo();
        return evaluationInfo;
    }

    public Mario getMario()
    {
        return levelScene.mario;
    }

    public int getTick()
    {
        return levelScene.tickCount;
    }

    public int getLevelDifficulty()
    {
        return levelScene.getLevelDifficulty();
    }

    public long getLevelSeed()
    {
        return levelScene.getLevelSeed();
    }

    public int getLevelType()
    {
        return levelScene.getLevelType();
    }

    public int getKilledCreaturesTotal()
    {
        return levelScene.getKillsTotal();
    }

    public int getLevelLength()
    {
        return levelScene.getLevelLength();
    }

    public int getLevelHeight()
    {
        return levelScene.getLevelHeight();
    }

    public int getKilledCreaturesByFireBall()
    {
        return levelScene.getKillsByFire();
    }

    public int getKilledCreaturesByShell()
    {
        return levelScene.getKillsByShell();
    }

    public int getKilledCreaturesByStomp()
    {
        return levelScene.getKillsByStomp();
    }

    public int getTimeLeft()
    {
        return levelScene.getTimeLeft();
    }

    public Level getLevel()
    {
        return levelScene.level;
    }

    private void computeEvaluationInfo()
    {
        if (recorder != null)
            closeRecorder();
        // evaluationInfo.agentType = agent.getClass().getSimpleName();
    }

```

```

// evaluationInfo.agentName = agent.getName();
evaluationInfo.marioStatus = levelScene.getMarioStatus();
evaluationInfo.flowersDevoured = Mario.flowersDevoured;
evaluationInfo.distancePassedPhys = (int) levelScene.mario.x;
evaluationInfo.distancePassedCells = levelScene.mario.mapX;
// evaluationInfo.totalLengthOfLevelCells = levelScene.level.getWidthCells();
// evaluationInfo.totalLengthOfLevelPhys = levelScene.level.getWidthPhys();
evaluationInfo.timeSpent = levelScene.getTimeSpent();
evaluationInfo.timeLeft = levelScene.getMarioStatus() == 1 ? levelScene.getTimeLeft() : 0;
evaluationInfo.coinsGained = Mario.coins;
evaluationInfo.totalNumberOfCoins = levelScene.level.counters.coinsCount;
evaluationInfo.totalNumberOfHiddenBlocks = levelScene.level.counters.hiddenBlocksCount;
evaluationInfo.totalNumberOfFlowers = levelScene.level.counters.flowers;
evaluationInfo.totalNumberOfMushrooms = levelScene.level.counters.mushrooms;
evaluationInfo.totalNumberOfCreatures = levelScene.level.counters.creatures;
evaluationInfo.marioMode = levelScene.getMarioMode();
evaluationInfo.mushroomsDevoured = Mario.mushroomsDevoured;
evaluationInfo.killsTotal = levelScene.getKillsTotal();
evaluationInfo.killsByStomp = levelScene.getKillsByStomp();
evaluationInfo.killsByFire = levelScene.getKillsByFire();
evaluationInfo.killsByShell = levelScene.getKillsByShell();
evaluationInfo.hiddenBlocksFound = Mario.hiddenBlocksFound;
evaluationInfo.collisionsWithCreatures = Mario.collisionsWithCreatures;
evaluationInfo.Memo = levelScene.memo;
evaluationInfo.levelLength = levelScene.level.length;
evaluationInfo.marioTraceFileName = marioTraceFile;
evaluationInfo.marioTrace = levelScene.level.marioTrace;
evaluationInfo.greenMushroomsDevoured = Mario.greenMushroomsDevoured;
evaluationInfo.bytecodeInstructions = PunctualJudge.getCounter();
}

public void setAgent(Agent agent)
{
    this.agent = agent;
}

public int getIntermediateReward()
{
    // TODO: reward for coins, killed creatures, cleared dead-ends, bypassed gaps, hidden blocks found
    return levelScene.getBonusPoints();
}

public int[] getMarioEgoPos()
{
    return marioEgoPos;
}

public int[] getMarioMovement()
{
    float xm = levelScene.mario.x - levelScene.mario.xOld;
    float ym = levelScene.mario.y - levelScene.mario.yOld;
    marioMovement[0] = xm > 0 ? 1 : xm < 0 ? -1 : 0;
    marioMovement[1] = ym > 0 ? 1 : ym < 0 ? -1 : 0;
    return marioMovement;
}

public void closeRecorder()
{
    if (recorder != null)
    {
        try
        {
            // recorder.closeFile();
            recorder.closeRecorder(getTimeSpent());
            //recorder = null;
        } catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}

public int getTimeSpent()
{
    return levelScene.getTimeSpent();
}

public byte[] getScreenCapture()
{
    return null;
}

public void setReplayer(Replayer replayer)
{

```

```

        levelScene.setReplayer(replayer);
    }

    public void saveLastRun(String filename)
    {
        if (recorder != null && recorder.canSave())
        {
            try
            {
                recorder.saveLastRun(filename);
            } catch (IOException ex)
            {
                System.err.println("[Mario AI EXCEPTION] : Recording could not be saved.");
                ex.printStackTrace();
            }
        }
    }

    //public void setRecording(boolean isRecording)
    //{
    //    this.isRecording = isRecording;
    //}
}



---



/*
 * Copyright (c) 2009-2010, Sergey Karakovskiy and Julian Togelius
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 * Neither the name of the Mario AI nor the
 * names of its contributors may be used to endorse or promote products
 * derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 * IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package ch.idsia.benchmark.tasks;

import ch.idsia.agents.Agent;
import ch.idsia.benchmark.mario.engine.GlobalOptions;
import ch.idsia.benchmark.mario.environments.Environment;
import ch.idsia.benchmark.mario.environments.MarioEnvironment;
import ch.idsia.tools.EvaluationInfo;
import ch.idsia.tools.MarioAIOptions;
import ch.idsia.tools.punj.PunctualJudge;
import ch.idsia.utils.statistics.StatisticalSummary;

import java.util.Vector;

/**
 * Created by IntelliJ IDEA.
 * User: Sergey Karakovskiy,
 * sergey@idsia.ch
 * Date: Mar 14, 2010 Time: 4:47:33 PM
 */

public class BasicTask implements Task
{
    protected final static Environment environment = MarioEnvironment.getInstance();
    private Agent agent;
    protected MarioAIOptions options;
    private long COMPUTATION_TIME_BOUND = 42; // stands for prescribed FPS 24.
    private String name = getClass().getSimpleName();
    private EvaluationInfo evaluationInfo;

    private Vector<StatisticalSummary> statistics = new Vector<StatisticalSummary>();

```

```

public BasicTask(MarioAIOptions marioAIOptions)
{
    this.setOptionsAndReset(marioAIOptions);
}

/**
 * @param repetitionsOfSingleEpisode
 * @return boolean flag whether controller is disqualified or not
 */
public boolean runSingleEpisode(final int repetitionsOfSingleEpisode)
{
    long c = System.currentTimeMillis();
    for (int r = 0; r < repetitionsOfSingleEpisode; ++r)
    {
        this.reset();
        while (!environment.isLevelFinished())
        {
            environment.tick();
            if (!GlobalOptions.isGameplayStopped)
            {
                c = System.currentTimeMillis();
                agent.integrateObservation(environment);
                agent.giveIntermediateReward(environment.getIntermediateReward());

                boolean[] action = agent.getAction();
                if (System.currentTimeMillis() - c > COMPUTATION_TIME_BOUND)
                    return false;
            }
            // System.out.println("action = " + Arrays.toString(action));
            // environment.setRecording(GlobalOptions.isRecording);
            environment.performAction(action);
        }
        environment.closeRecorder(); //recorder initialized in environment.reset
        environment.getEvaluationInfo().setTaskName(name);
        this.evaluationInfo = environment.getEvaluationInfo().clone();
    }

    return true;
}

public Environment getEnvironment()
{
    return environment;
}

public int evaluate(Agent controller)
{
    return 0;
}

public void setOptionsAndReset(MarioAIOptions options)
{
    this.options = options;
    reset();
}

public void setOptionsAndReset(final String options)
{
    this.options.setArgs(options);
    reset();
}

public void doEpisodes(int amount, boolean verbose, final int repetitionsOfSingleEpisode)
{
    for (int j = 0; j < EvaluationInfo.numberElements; j++)
    {
        statistics.addElement(new StatisticalSummary());
    }
    for (int i = 0; i < amount; ++i)
    {
        this.reset();
        this.runSingleEpisode(repetitionsOfSingleEpisode);
        if (verbose)
            System.out.println(environment.getEvaluationInfoAsString());

        for (int j = 0; j < EvaluationInfo.numberElements; j++)
        {
            statistics.get(j).add(environment.getEvaluationInfoAsInts()[j]);
        }
    }

    System.out.println(statistics.get(3).toString());
}

```

```

public boolean isFinished()
{
    return false;
}

public void reset()
{
    agent = options.getAgent();
    environment.reset(options);
    agent.reset();
    agent.setObservationDetails(environment.getReceptiveFieldWidth(),
        environment.getReceptiveFieldHeight(),
        environment.getMarioEgoPos()[0],
        environment.getMarioEgoPos()[1]);
}

public String getName()
{
    return name;
}

public void printStatistics()
{
    System.out.println(evaluationInfo.toString());
}

public EvaluationInfo getEvaluationInfo()
{
    // System.out.println("evaluationInfo = " + evaluationInfo);
    return evaluationInfo;
}

// start timer
// long tm = System.currentTimeMillis();

// System.out.println("System.currentTimeMillis() - tm > COMPUTATION_TIME_BOUND = " + (System.currentTimeMillis() - tm));
// if (System.currentTimeMillis() - tm > COMPUTATION_TIME_BOUND)
// {
//     # controller disqualified on this level
//     System.out.println("Agent is disqualified on this level");
//     return false;
// }



---


/*
 * Copyright (c) 2009-2010, Sergey Karakovskiy and Julian Togelius
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 * Neither the name of the Mario AI nor the
 * names of its contributors may be used to endorse or promote products
 * derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 * IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

package ch.idsia.benchmark.mario.engine.level;

import bsh.This;
import ch.idsia.benchmark.mario.engine.sprites.Sprite;
import ch.idsia.tools.MarioAIOptions;
import ch.idsia.tools.RandomCreatureGenerator;
import ch.idsia.utils.ErrorCodes;

```

```

import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.Random;

/**
 * This class is simple to use. Just call <b>createLevel</b> method with params:
 * <ul>
 * <li>MarioAIOptions args, that contains: ... TODO:TASK:[M]
 * </li>
 * <li>length -- length of the level in cells. One cell is 16 pixels long</li>
 * <li>height -- height of the level in cells. One cell is 16 pixels long </li>
 * <li>seed -- use this param to make a globalRandom level.
 * <li>On different machines with the same seed param there will be one level</li>
 * <li>levelDifficulty -- use this param to change difficult of the level.
 * <li>On different machines with the same seed param there will be one level</li>
 * <li>levelType -- levelType of the level. One of Overground, Underground, Castle.</li>
 * </ul>
 *
 * @see #TYPE_OVERGROUND
 * @see #TYPE_UNDERGROUND
 * @see #TYPE_CASTLE
 */

public class LevelGenerator
{
    public static final int TYPE_OVERGROUND = 0;
    public static final int TYPE_UNDERGROUND = 1;
    public static final int TYPE_CASTLE = 2;

    public static final int DEFAULT_FLOOR = -1;

    public static final int LevelLengthMinThreshold = 50; // minimal length of the level. used in ToolsConfigurator
    private static boolean isFlatLevel;

    private static int length;
    private static int height;
    private static Level level;

    private static Random globalRandom = new Random(0);
    private static Random ceilingRandom = new Random(0);
    private static RandomCreatureGenerator creaturesRandom = new RandomCreatureGenerator(0, "", 0);
    public static Random dxRnd = new Random(0); //used in addEnemy to compute dx

    private static final int ODDS_STRAIGHT = 0;
    private static final int ODDS_HILL_STRAIGHT = 1;
    private static final int ODDS_TUBES = 2;
    private static final int ODDS_GAPS = 3;
    private static final int ODDS_CANNONS = 4;
    private static final int ODDS_DEAD_ENDS = 5;
    private static int[] odds = new int[6];
    private static int totalOdds;
    private static int levelDifficulty;
    private static int levelType;
    private static int levelSeed;

    private static boolean isLadder = false;

    private static final int ANY_HEIGHT = -1;
    private static final int INFINITE_FLOOR_HEIGHT = Integer.MAX_VALUE;

    //Level customization counters
    static Level.objCounters counters = new Level.objCounters();

    private LevelGenerator() {}

    private static void loadLevel(String filePath)
    {
        try
        {
            if (filePath.equals("")) //This must never happen
            {
                System.err.println("[MarioAI ERROR] : level file path is empty; exiting...");
                System.exit(ErrorCodes.FILE_NAME_OR_LOAD_PROBLEM);
            }

            level = Level.load(new ObjectInputStream(new FileInputStream(filePath)));
        } catch (IOException e)
        {
            System.err.println("[MarioAI EXCEPTION] : failed while trying to loadAgent " + filePath);
            System.exit(ErrorCodes.FILE_NAME_OR_LOAD_PROBLEM);
        } catch (ClassNotFoundException e)
        {
            System.err.println("[MarioAI EXCEPTION] : class not found in " + filePath);
        }
    }

```

```

        System.exit(ErrorCodes.FILE_NAME_OR_LOAD_PROBLEM);
    }
}

public static Level createLevel(MarioAIOptions args)
{
    // -ls option can also loadAgent level from file if filename instead of a number provided
    levelType = args.getLevelType();
    try
    {
        levelSeed = args.getLevelRandSeed() + levelType;
    } catch (Exception e)
    {
        loadLevel(args.getParameterValue("-ls"));
        return level;
    }
    length = args.getLevelLength();
    height = args.getLevelHeight();
    if (height < 15)
    {
        System.err.println("[Mario AI WARNING] : Level height changed to minimal allowed value 15");
        height = 15;
    }
    isFlatLevel = args.isFlatLevel();

    counters.reset(args);
    levelDifficulty = args.getLevelDifficulty();
    odds[ODDS_STRAIGHT] = 20;
    odds[ODDS_HILL_STRAIGHT] = 1;
    odds[ODDS_TUBES] = 2 + 1 * levelDifficulty;
    odds[ODDS_GAPS] = 3 * levelDifficulty;
    odds[ODDS_CANNONS] = -10 + 5 * levelDifficulty;
    odds[ODDS_DEAD_ENDS] = 2 + 2 * levelDifficulty;

    if (levelType != LevelGenerator.TYPE_OVERGROUND)
        odds[ODDS_HILL_STRAIGHT] = 0; //no hill straight in TYPE_OVERGROUND level

    totalOdds = 0;
    for (int i = 0; i < odds.length; i++)
    {
        if (odds[i] < 0) odds[i] = 0;
        totalOdds += odds[i];
        odds[i] = totalOdds - odds[i];
    }
    if (totalOdds <= 0)
    {
        System.err.println("[Mario AI SURPRISE] : UNEXPECTED level will be generated");
        totalOdds = 1;
    }

    level = new Level(length, height);
    // levelSeed = args.getLevelRandSeed();// + levelType; // TODO:TASK:[M] ensure the difference of underground,
    castle
    globalRandom.setSeed(levelSeed);
    creaturesRandom.setSeed(levelSeed, args.getEnemies(), levelDifficulty);
    ceilingRandom.setSeed(levelSeed);
    dxRnd.setSeed(levelSeed);

    isLadder = args.isLevelLadder();

    int currentLength = 0; //total level currentLength so far

    //by default mario supposed to start on a straight surface
    int floor = DEFAULT_FLOOR;
    if (isFlatLevel)
        floor = height - 1 - globalRandom.nextInt(4);

    currentLength += buildStraight(0, level.length, true, floor, INFINITE_FLOOR_HEIGHT);
    while (currentLength < level.length - 10)
    {
        // System.out.println("level.currentLength - currentLength = " + (level.currentLength - currentLength));
        currentLength += buildZone(currentLength, level.length - currentLength, ANY_HEIGHT, floor,
            INFINITE_FLOOR_HEIGHT);
    }

    if (!isFlatLevel) //NOT flat level
        floor = height - 1 - globalRandom.nextInt(4); //floor of the exit line

    //coordinates of the exit
    level.xExit = args.getExitX();
    level.yExit = args.getExitY();

    if (level.xExit == 0)
        level.xExit = level.length - 1;
}

```



```

        if (level.yExit == 0)
            level.yExit = floor - 1;

        level.randomSeed = levelSeed;
        level.type = levelType;
        level.difficulty = levelDifficulty;

        //level zone where exit is located
        for (int x = currentLength; x < level.length; x++)
        {
            for (int y = 0; y < height; y++)
            {
                if (y >= floor)
                {
                    level.setBlock(x, y, (byte) (1 + 9 * 16));
                }
            }
        }
        // level.setSpriteTemplate(level.xExit, level.yExit, new SpriteTemplate(Sprite.KIND_PRINCESS));

        // //if underground or castle then build ceiling
        // if (levelType == LevelGenerator.TYPE_CASTLE || levelType == LevelGenerator.TYPE_UNDERGROUND)
        // {
        //     int ceiling = 0;
        //     int run = 0;
        //     for (int x = 0; x < level.length; x++)
        //     {
        //         if (run-- <= 0 && x > 4)
        //         {
        //             ceiling = globalRandom.nextInt(4);
        //             run = globalRandom.nextInt(4) + 4;
        //         }
        //         for (int y = 0; y < level.height; y++)
        //         {
        //             if ((x > 4 && y <= ceiling) || x < 1)
        //             {
        //                 level.setBlock(x, 0, (byte) (1 + 9 * 16));
        //             }
        //         }
        //     }
        // }

        fixWalls();
        setPrincess(level.xExit, level.yExit);

        level.counters = counters;

        return level;
    }

    private static void setPrincess(int x, int y)
    {
        // System.out.println("x = " + x);
        // System.out.println("y = " + y);
        level.setSpriteTemplate(x, y, new SpriteTemplate(Sprite.KIND_PRINCESS));
        level.setBlock(x, y, (byte) (15 + 15 * 16));
    }

    private static int buildZone(int x, int maxLength, int maxHeight, int floor, int floorHeight)
    {
        // System.out.println("buildZone maxLength = " + maxLength);
        int t = globalRandom.nextInt(totalOdds);
        int type = 0;
        //calculate what will be built
        for (int i = 0; i < odds.length; i++)
        {
            if (odds[i] <= t)
            {
                type = i;
            }
        }

        int length = 0;

        switch (type)
        {
            case ODDS_STRAIGHT:
                length = buildStraight(x, maxLength, false, floor, floorHeight);
                break;
            case ODDS_HILL_STRAIGHT:
                if (floor == DEFAULT_FLOOR && counters.hillStraightCount < counters.totalHillStraight)
                {

```

```

        counters.hillStraightCount++;
// length = buildHillStraight(x, maxLength, floor, false);
        length = buildHill(x, true, maxLength, floor, false);
    } else
        length = 0;
    break;
case ODDS_TUBES:
    if (counters.tubesCount < counters.totalTubes)
        length = buildTubes(x, maxLength, maxHeight, floor, floorHeight);
    else
        length = 0;
    break;
case ODDS_GAPS:
    if ((floor > 2 || floor == ANY_HEIGHT) && (counters.gapsCount < counters.totalGaps))
    {
        counters.gapsCount++;
        length = buildGap(x, maxLength, maxHeight, floor, floorHeight);
    } else
        length = 0;
    break;
case ODDS_CANNONS:
    if (counters.cannonsCount < counters.totalCannons)
        length = buildCannons(x, maxLength, maxHeight, floor, floorHeight);
    else
        length = 0;
    break;
case ODDS_DEAD_ENDS:
    {
        if (floor == DEFAULT_FLOOR && counters.deadEndsCount < counters.totalDeadEnds) //if method was not
            called from buildDeadEnds
        {
            counters.deadEndsCount++;
            length = buildDeadEnds(x, maxLength);
        }
    }
}

/**
 * EEdit by montywest
 *
 * Suspected bug, previously was added enemies to 0 length zones
 * New algorithm scales enemies more gently
 */
addZonalEnemies(length, x);

if (levelType > 0)
    buildCeiling(x, length);

return length;
}

/**
 * @author montywest
 * Public for testing
 */
public static void addZonalEnemies(int zoneLength, int zoneStart) {
    if (zoneLength != 0) {

        int heads = 0;
        int flips = zoneLength - 1;
        int chance = (int)2.5d*(levelDifficulty+4);
        for (int flip = 0; flip < flips; flip++) {
            if (chance > creaturesRandom.nextInt(100)) {
                heads++;
            }
        }

        int crCount = 0;
        int tries = 0;
        int yy = level.height;
        while(crCount != heads && tries < 50) {
            int yTries = 0;
            boolean added = false;
            while (!added && yTries < 4) {
                int dx = creaturesRandom.nextInt(zoneLength);
                int xx = zoneStart+dx;
                if (level.getBlock(xx, yy) == 0 &&
                    level.getSpriteTemplate(xx, yy) == null) {
                    addEnemy(xx, yy);
                    ++crCount;
                    added = true;
                } else {

```

```

        }
        yy--;
        tries++;
    }

    // for (int yy = level.height; yy > 0; yy--)
    // if (level.getBlock(x, yy) == 0 &&
    // creaturesRandom.nextInt(levelDifficulty + 1) + 1 > (levelDifficulty + 1) / 2 &&
    // crCount < levelDifficulty + 1 &&
    // level.getSpriteTemplate(x, yy) == null)
    // {
    //     addEnemy(x, yy);
    //     ++crCount;
    // }
    // System.out.println("Length-" + length + " Enemies-" + crCount);
}

private static void buildCeiling(int x0, int length)
{
    int maxCeilingHeight = 3;
    int ceilingLength = length;

    if (ceilingLength < 2)
        return;

    // len:
    // for (int i = x0; i < x1; i++)
    // for (int j = 0; j < height; j++)
    // if (level.getBlock(i, j) != 0)
    // {
    //     maxCeilingHeight = j;
    //     break len;
    // }
    int len = 0;

    while (len < ceilingLength)
    {
        int sectionLength = ceilingRandom.nextInt(2) + 2;
        if (sectionLength > ceilingLength)
            sectionLength = ceilingLength;

        // if (maxCeilingHeight > 0)
        // maxCeilingHeight--;
        // if (maxCeilingHeight == 0)
        // maxCeilingHeight = 1;
        // if (maxCeilingHeight > 5)
        // maxCeilingHeight = 5;

        int height = ceilingRandom.nextInt(maxCeilingHeight) + 1;
        for (int i = 0; i < sectionLength; i++)
        {
            for (int j = 0; j < height; j++)
                level.setBlock(x0 + len + i, j, (byte) (1 + 9 * 16));
        }

        len += sectionLength;
    }
}

private static void addEnemy(int x, int y)
{
    if (!creaturesRandom.canAdd())
        return;

    int dx = (int) dxRnd.nextGaussian();
    int creatureKind = creaturesRandom.nextCreature();
    if (creatureKind != Sprite.KIND_UNDEF)
    {
        if (level.setSpriteTemplate(x + dx, y, new SpriteTemplate(creatureKind)))
            ++counters.creatures;
        else
            creaturesRandom.increaseLastCreature();
    }
}

//x0 - first block to start from
//maxLength - maximal length of the zone

private static int buildDeadEnds(int x0, int maxLength)
{
    //first of all build pre dead end zone
    int floor = height - 2 - globalRandom.nextInt(2); //floor of pre dead end zone

```

```

int length = 0; // total zone length
int preDeadEndLength = 7 + globalRandom.nextInt(10);
int rHeight = floor - 1; //rest height
int separatorY = 3 + globalRandom.nextInt(rHeight - 7); //Y coordinate of the top line of the separator

length += buildStraight(x0, preDeadEndLength, true, floor, INFINITE_FLOOR_HEIGHT); //buildZone( x0, x0+
preDeadEndLength, floor ); //build pre dead end zone

if (globalRandom.nextInt(3) == 0 && isLadder)
{
    int ladderX = x0 + globalRandom.nextInt(length - 1) + 1;
    if (ladderX > x0 + length)
        ladderX = x0 + length;
    buildLadder(ladderX, floor, floor - separatorY);
} else
    buildBlocks(x0, x0 + preDeadEndLength, floor, true, 0, 0, true, true);

//correct direction
//true - top, false = bottom
globalRandom.nextInt();
int k = globalRandom.nextInt(5); //(globalRandom.nextInt() % (this.levelDifficulty+1));
boolean direction = globalRandom.nextInt(k + 1) != 1;

//Y coordinate of the bottom line of the separator is determined as separatorY + separatorHeight
int separatorHeight = 2 + globalRandom.nextInt(2);

int nx = x0 + length;
int depth = globalRandom.nextInt(levelDifficulty + 1) + 2 * (1 + levelDifficulty);
if (depth + length > maxLength)
{
    // depth = maxLength
    while (depth + length > maxLength - 1)
    {
        depth--;
    }
}

int tLength = 0;
int bSpace = floor - (separatorY + separatorHeight);
if (bSpace < 4)
{
    while (bSpace < 4)
    {
        separatorY -= 1;
        bSpace = floor - (separatorY + separatorHeight);
    }
}

int wallWidth = 2 + globalRandom.nextInt(3);

while (tLength < depth) //top part
{
    tLength += buildZone(nx + tLength, depth - tLength, separatorY - 1, separatorY, separatorHeight);
}
tLength = 0;
while (tLength < depth) //bottom part
{
    tLength += buildZone(nx + tLength, depth - tLength, bSpace, floor, INFINITE_FLOOR_HEIGHT);
}

boolean wallFromBlocks = false; //globalRandom.nextInt(5) == 2;

for (int x = nx; x < nx + depth; x++)
{
    for (int y = 0; y < height; y++)
    {
        if (x - nx >= depth - wallWidth)
        {
            if (direction) //wall on the top
            {
                if (y <= separatorY // + separatorHeight )
                {
                    if (wallFromBlocks)
                        level.setBlock(x, y, (byte) (0 + 1 * 16));
                    else
                        level.setBlock(x, y, (byte) (1 + 9 * 16));
                }
            } else
            {
                if (y >= separatorY)
                {
                    if (wallFromBlocks)
                        level.setBlock(x, y, (byte) (0 + 1 * 16));
                    else

```

```

        level.setBlock(x, y, (byte) (1 + 9 * 16));
    }
}

return length + tLength;
}

private static void buildLadder(int x0, int floor, int maxHeight)
{
    int ladderHeight = globalRandom.nextInt(height);
    if (ladderHeight > maxHeight && maxHeight != ANY_HEIGHT)
    {
        ladderHeight = maxHeight;
    }

    if (ladderHeight < 4)
        return;

    for (int y = floor, i = 0; i < ladderHeight - 1; y--, i++)
        level.setBlock(x0, y - 1, (byte) (13 + 3 * 16));

    level.setBlock(x0, floor - ladderHeight, (byte) (13 + 5 * 16));
}

/**
 * @author montywest
 * Public for testing
 */
public static int getGapLength(int ld) {
    double m = 0.02d;
    double a = 0.4d;
    int maxL = 3;
    if (ld >= 0 && ld <= 3)
        maxL = 3;
    else if (ld <= 6)
        maxL = 4;
    else if (ld <= 10)
        maxL = 6;
    else if (ld <= 15)
        maxL = 9;
    else
        maxL = 10;

    int calcL = 2 + ((int) Math.ceil(globalRandom.nextInt(ld+1) * ((m*ld) + a)));
    return Math.min(maxL, calcL);
}

private static int buildGap(int xo, int maxLength, int maxHeight, int vfloor, int floorHeight)
{
    int gs = globalRandom.nextInt(5) + 2; //GapStairs
    int gl = getGapLength(levelDifficulty); //GapLength
    // System.out.println("LD: " + levelDifficulty + " - GapLength: " + gl);
    int length = gs * 2 + gl;

    if (length > maxLength)
        length = maxLength;

    boolean hasStairs = globalRandom.nextInt(3) == 0;
    if (isFlatLevel || (maxHeight <= 5 && maxHeight != ANY_HEIGHT))
    {
        hasStairs = false;
    }

    int floor = vfloor;
    if (vfloor == DEFAULT_FLOOR && !isFlatLevel)
    {
        floor = height - 1 - globalRandom.nextInt(4);
    } else //code in this block is a magic. don't change it
    {
        floor++;
    }
    // globalRandom.nextInt();
    if (floor > 1)
    {
        floor -= 1;
    }

    if (floorHeight == INFINITE_FLOOR_HEIGHT)
    {
        floorHeight = height - floor;
    }
}

```

```

//
// if (gs > 3 && creaturesRandom.nextInt(35) > levelDifficulty + 1 && !hasStairs)
// {
//     addEnemiesLine(xo, xo + gs - 1, floor - 1);
//     addEnemiesLine(xo + length - gs, xo + length - 1, floor - 1);
// }

for (int x = xo; x < xo + length; x++)
{
    if (x < xo + gs || x > xo + length - gs - 1)
    {
        for (int y = 0; y < height; y++)
        {
            if (y >= floor && y <= floor + floorHeight)
                level.setBlock(x, y, (byte) (1 + 9 * 16));
            else if (hasStairs)
            {
                if (x < xo + gs)
                {
                    if (y >= floor - (x - xo) + 1 && y <= floor + floorHeight)
                        level.setBlock(x, y, (byte) (9 + 0 * 16));
                } else if (y >= floor - ((xo + length) - x) + 2 && y <= floor + floorHeight)
                    level.setBlock(x, y, (byte) (9 + 0 * 16));
            }
        }
    }
}
if (gl > 8)
{
    buildHill(xo + gs + globalRandom.nextInt(Math.abs((gl - 4)) / 2 + 1), false, 3, floor, true);
}

return length;
}

private static int buildCannons(int xo, int maxLength, int maxHeight, int vfloor, int floorHeight)
{
    int maxCannonHeight = 0;
    int length = globalRandom.nextInt(10) + 2;
    if (length > maxLength) length = maxLength;

    int floor = vfloor;
    if (vfloor == DEFAULT_FLOOR)
    {
        floor = height - 1 - globalRandom.nextInt(4);
    } else
    {
        globalRandom.nextInt();
    }

    if (floorHeight == INFINITE_FLOOR_HEIGHT)
    {
        floorHeight = height - floor;
    }

    int oldXCannon = -1;

    int xCannon = xo + 1 + globalRandom.nextInt(4);
    for (int x = xo; x < xo + length; x++)
    {
        if (x > xCannon)
        {
            xCannon += 2 + globalRandom.nextInt(4);
            counters.cannonsCount++;
        }
        if (xCannon == xo + length - 1)
        {
            xCannon += 10;
        }

        int cannonHeight = floor - globalRandom.nextInt(3) - 1; //cannon height is a Y coordinate of top part of
        the cannon
        if (maxHeight != ANY_HEIGHT)
        {
            //maxHeight -= 2;
            if (floor - cannonHeight >= maxHeight)
            {
                if (maxHeight > 4)
                {
                    maxHeight = 4;
                }
                while (floor - cannonHeight > maxHeight)
                {

```

```

        cannonHeight++;
    }
}
if (cannonHeight > maxCannonHeight)
    maxCannonHeight = cannonHeight;
}

for (int y = 0; y < height; y++)
{
    if (y >= floor && y <= floor + floorHeight)
    {
        level.setBlock(x, y, (byte) (1 + 9 * 16));
    } else if (counters.cannonsCount <= counters.totalCannons)
    {
        if (x == xCannon && y >= cannonHeight && y <= floor) // + floorHeight
        {
            if (y == cannonHeight)
            {
                if (oldXCannon != -1 && creaturesRandom.nextInt(35) > levelDifficulty + 1)
                {
                    // addEnemiesLine(oldXCannon + 1, xCannon - 1, floor - 1);
                }
                oldXCannon = x;
                level.setBlock(x, y, (byte) (14 + 0 * 16)); // cannon barrel
            } else if (y == cannonHeight + 1)
            {
                level.setBlock(x, y, (byte) (14 + 1 * 16)); // base for cannon barrel
            } else
            {
                level.setBlock(x, y, (byte) (14 + 2 * 16)); // cannon pole
            }
        }
    }
}

if (globalRandom.nextBoolean())
    buildBlocks(xo, xo + length, floor - maxCannonHeight - 2, false, 0, 0, false, false);

return length;
}

private static int buildHill(int x0, boolean withStraight, int maxLength, int vfloor, boolean isInGap)
{
    int length = globalRandom.nextInt(10) + 10;
    if (length > maxLength)
    {
        length = maxLength;
    }

    int floor = vfloor;
    if (vfloor == DEFAULT_FLOOR)
    {
        floor = height - 1 - globalRandom.nextInt(4);
    }

    if (withStraight)
    {
        for (int x = x0; x < x0 + length; x++)
        {
            for (int y = 0; y < height; y++)
            {
                if (y >= floor)
                {
                    level.setBlock(x, y, (byte) (1 + 9 * 16));
                }
            }
        }
    }

    boolean canBuild = true;

    int top = floor;
    if (isInGap)
        floor = level.height;
    while (canBuild)
    {
        top -= isFlatLevel ? 0 : (globalRandom.nextInt(2) + 2);
        if (top < 0)
            canBuild = false;
        else
        {
            int l = globalRandom.nextInt(length / 2) + 1;
            int xx0 = globalRandom.nextInt(l + 1) + x0;

```

```

        if (globalRandom.nextInt(4) == 0)
        {
            decorate(xx0 - 1, xx0 + 1 + 1, top);
            canBuild = false;
        }
        for (int x = xx0; x < xx0 + 1; x++)
        {
            for (int y = top; y < floor; y++)
            {
                int xx = 5;
                if (x == xx0) xx = 4;
                if (x == xx0 + 1 - 1) xx = 6;
                int yy = 9;
                if (y == top) yy = 8;

                if (level.getBlock(x, y) == 0)
                {
                    level.setBlock(x, y, (byte) (xx + yy * 16));
                } else
                {
                    if (level.getBlock(x, y) == (byte) (4 + 8 * 16))
                        level.setBlock(x, y, (byte) (4 + 11 * 16));
                    if (level.getBlock(x, y) == (byte) (6 + 8 * 16))
                        level.setBlock(x, y, (byte) (6 + 11 * 16));
                }
            }
        }
    }
    // addEnemy(xx0, top - 1);
}

return length;
}

private static int buildTubes(int xo, int maxLength, int maxHeight, int vfloor, int floorHeight)
{
    int maxTubeHeight = 0;
    int length = globalRandom.nextInt(10) + 5;
    if (length > maxLength) length = maxLength;

    int floor = vfloor;
    if (vfloor == DEFAULT_FLOOR)
    {
        floor = height - 1 - globalRandom.nextInt(4);
    } else
    {
        globalRandom.nextInt();
    }
    int xTube = xo + 1 + globalRandom.nextInt(4);

    int tubeHeight = floor - globalRandom.nextInt(3) - 1;

    if (maxHeight != ANY_HEIGHT)
    {
        //maxHeight -= 2;
        if (floor - tubeHeight > maxHeight)
        {
            if (maxHeight > 4)
            {
                maxHeight = 4;
            }
            while (floor - tubeHeight > maxHeight)
            {
                tubeHeight++;
            }
        }
    }

    if (floorHeight == INFINITE_FLOOR_HEIGHT)
    {
        floorHeight = height - floor;
    }

    int oldXTube = -1;

    for (int x = xo; x < xo + length; x++)
    {
        if (x > xTube + 1)
        {
            xTube += 3 + globalRandom.nextInt(4);
            tubeHeight = floor - globalRandom.nextInt(2) - 2;
            if (maxHeight != ANY_HEIGHT)
            {

```



```

        while (floor - tubeHeight > maxHeight - 1)
        {
            tubeHeight++;
        }
    }

    if (tubeHeight > maxTubeHeight)
        maxTubeHeight = tubeHeight;
}
if (xTube >= xo + length - 2)
{
    xTube += 10;
}

if (x == xTube && globalRandom.nextInt(11) < levelDifficulty + 1 && creaturesRandom.isCreatureEnabled("f"))
{
    level.setSpriteTemplate(x, tubeHeight, new SpriteTemplate(Sprite.KIND_ENEMY_FLOWER));
    ++counters.creatures;
}

for (int y = 0; y < floor + floorHeight; y++)
{
    if (y >= floor && y <= floor + floorHeight)
        level.setBlock(x, y, (byte) (1 + 9 * 16));
    else
    {
        if ((x == xTube || x == xTube + 1) && y >= tubeHeight)
        {
            int xPic = 10 + x - xTube;
            if (y == tubeHeight)
            {
                level.setBlock(x, y, (byte) (xPic + 0 * 16));
                if (x == xTube)
                {
                    if (oldXTube != -1 && creaturesRandom.nextInt(35) > levelDifficulty + 1)
                    {
                        // addEnemiesLine(oldXTube + 2, xTube - 1, floor - 1);
                    }
                    oldXTube = x;
                    counters.tubesCount++;
                }
            } else
            {
                level.setBlock(x, y, (byte) (xPic + 1 * 16));
            }
        }
    }
}

if (globalRandom.nextBoolean())
    buildBlocks(xo, xo + length, floor - maxTubeHeight - 2, false, 0, 0, false, false);

return length;
}

// parameter safe should be set to true iff length of the Straight > 10.
// minimal length = 2
// floorHeight - height of the floor. used for building of the top part of the dead end separator

private static int buildStraight(int xo, int maxLength, boolean safe, int vfloor, int floorHeight)
{
    int length;
    if (floorHeight != INFINITE_FLOOR_HEIGHT)
    {
        length = maxLength;
    } else
    {
        length = globalRandom.nextInt(8) + 2; // globalRandom.nextInt(50)+1 + 2;
        if (safe) length = 10 + globalRandom.nextInt(5);
        if (length > maxLength) length = maxLength;
    }

    int floor = vfloor;
    if (vfloor == DEFAULT_FLOOR)
    {
        floor = height - 1 - globalRandom.nextInt(4);
    } else
    {
        globalRandom.nextInt();
    }
}

```

```

    int y1 = height;
    if (floorHeight != INFINITE_FLOOR_HEIGHT)
    {
        y1 = floor + floorHeight;
    }

    for (int x = xo; x < xo + length; x++)
        for (int y = floor; y < y1; y++)
            if (y >= floor)
                level.setBlock(x, y, (byte) (1 + 9 * 16));

    if (!safe)
    {
        if (length > 5)
        {
            decorate(xo, xo + length, floor);
        }
    }

    return length;
}

private static boolean canBuildBlocks(int x0, int floor, boolean isHB)
{
    if ((counters.blocksCount >= counters.totalBlocks && !isHB))
    {
        return false;
    }

    boolean res = true;

    // if (floor < 1)
    // {
    //     return false;
    // }

    // for (int y = 0; y < 1; y++)
    // {
    //     if (level.getBlock(x0, floor - y) != 0)
    //     {
    //         res = false;
    //         break;
    //     }
    // }

    return res;
}

private static boolean buildBlocks(int x0, int x1, int floor, boolean pHB, int pS, int pE, boolean onlyHB,
    boolean isDistance)
{
    boolean result = false;
    if (counters.blocksCount > counters.totalBlocks)
    {
        return false;
    }
    int s = pS; //Start
    int e = pE; //End
    boolean hb = pHB;

    if (onlyHB)
        hb = onlyHB;

    --floor;
    while (floor > 0) //minimal distance between the bricks line and floor is 4
    {
        if ((x1 - 1 - e) - (x0 + 1 + s) > 0) //minimal number of bricks in the line is positive value
        {
            for (int x = x0 + s; x < x1 - e; x++)
            {
                if (hb && counters.totalHiddenBlocks != 0) //if hidden blocks to be built
                {
                    boolean isBlock = globalRandom.nextInt(2) == 1;
                    if (isBlock && canBuildBlocks(x, floor - 4, true))
                    {
                        level.setBlock(x, floor - 4, (byte) (1)); //a hidden block with a coin
                        counters.hiddenBlocksCount++;
                        ++counters.coinsCount;
                    }
                } else
                {
                    boolean canDeco = false; //can add enemy line and coins
                    //decorate( x0, x1, floor, true );
                }
            }
        }
        floor--;
    }
}

```

```

if (x != x0 + 1 && x != x1 - 2 && globalRandom.nextInt(3) == 0)
{
    if (canBuildBlocks(x, floor - 4, false))
    {
        counters.blocksCount++;
        int rnd = globalRandom.nextInt(6);
        if (rnd >= 0 && rnd < 2)
        {
            if (level.getBlock(x, floor) == 0)
                level.setBlock(x, floor, (byte) (4 + 2 + 1 * 16)); //a brick with animated
                question symbol with power up. when broken becomes a rock
        } else if (rnd >= 2 && rnd < 4)
        {
            if (level.getBlock(x, floor) == 0)
            {
                level.setBlock(x, floor, (byte) (4 + 1 + 1 * 16)); //a brick with animated
                question symbol with coin. when broken becomes a rock
                ++counters.coinsCount;
            }
        } else if (rnd >= 4 && rnd < 6)
        {
            int coinsNumber = globalRandom.nextInt(9) + 1;
            level.setBlock(x, floor, (byte) (4 + 3 + 1 * 16)); //a brick with animated question
            symbol with N coins inside. when broken becomes a rock
            level.setBlockData(x, floor, (byte) -coinsNumber);
            counters.coinsCount += coinsNumber;
        }
        canDeco = true;
    }
} else if (globalRandom.nextInt(4) == 0)
{
    if (canBuildBlocks(x, floor - 4, false))
    {
        counters.blocksCount++;
        if (globalRandom.nextInt(4) == 0)
        {
            if (level.getBlock(x, floor) == 0)
                level.setBlock(x, floor, (byte) (2 + 1 * 16)); //a brick with a power up. when
                broken becomes a rock
        } else
        {
            if (level.getBlock(x, floor) == 0)
            {
                level.setBlock(x, floor, (byte) (1 + 1 * 16)); //a brick with a coin. when
                broken becomes a rock
                ++counters.coinsCount;
            }
        }
        canDeco = true;
    }
} else if (globalRandom.nextInt(2) == 1 && canBuildBlocks(x, floor - 4, false))
{
    if (level.getBlock(x, floor) == 0)
    {
        counters.blocksCount++; //TODO:TASK:!!H! move it in to the Level.setBlock
        level.setBlock(x, floor, (byte) (0 + 1 * 16)); //a break brick
        canDeco = true;
    }
}
if (canDeco)
{
    //if (creaturesRandom.nextInt(35) > levelDifficulty + 1) addEnemiesLine(x0 + 1, x1 - 1,
    floor - 1);
    buildCoins(x0, x1, floor, s, e);
}
}
}
if (onlyHB)
{
    hb = true;
} else
{
    hb = globalRandom.nextInt(4) == 0; //globalRandom.nextInt(3) == globalRandom.nextInt(3);
}
}

// if (creaturesRandom.nextInt(35) > levelDifficulty + 1)
// addEnemiesLine(x0 + 1, x1 - 1, floor - 1);

int delta = isDistance ? 4 : globalRandom.nextInt(6) + 3;
if (delta > 4)
    result = true;
floor -= delta;
s = globalRandom.nextInt(4);

```

```

        e = globalRandom.nextInt(4);
    }
    globalRandom.nextBoolean();
    return result;
}

private static void buildCoins(int x0, int x1, int floor, int s, int e)
{
    if (floor - 2 < 0) return;
    //if (!isFlatLevel)
    if ((x1 - 1 - e) - (x0 + 1 + s) > 1)
    {
        for (int x = x0 + 1 + s; x < x1 - 1 - e; x++)
        {
            if (counters.coinsCount >= counters.totalCoins)
            {
                break;
            }
            if (level.getBlock(x, floor - 2) == 0)
            {
                counters.coinsCount++;
                level.setBlock(x, floor - 2, (byte) (2 + 2 * 16)); //coin
            }
        }
    }
}

private static void decorate(int x0, int x1, int floor)
{
    if (floor < 1) return;

    int s = globalRandom.nextInt(4);
    int e = globalRandom.nextInt(4);
    boolean hb = ((globalRandom.nextInt(levelDifficulty + 1) % (levelDifficulty + 1))) > 0.5;

    // if (!hb)
    // {
    //     addEnemiesLine(x0 + 1, x1 - 1, floor - 1);
    // }

    if (floor - 2 > 0 && !hb)
    {
        buildCoins(x0, x1, floor, s, e);
    }

    boolean buildLadder = buildBlocks(x0, x1, floor, hb, s, e, false, false);

    if (buildLadder && isLadder && globalRandom.nextInt(3) == 0)
        buildLadder(globalRandom.nextBoolean() ? x0 : x1, floor, ANY_HEIGHT);
}

private static void fixWalls()
{
    boolean[][] blockMap = new boolean[length + 1][height + 1];
    for (int x = 0; x < length + 1; x++)
    {
        for (int y = 0; y < height + 1; y++)
        {
            int blocks = 0;
            for (int xx = x - 1; xx < x + 1; xx++)
            {
                for (int yy = y - 1; yy < y + 1; yy++)
                {
                    if (level.getBlockCapped(xx, yy) == (byte) (1 + 9 * 16)) blocks++;
                }
            }
            blockMap[x][y] = blocks == 4;
        }
    }
    blockify(level, blockMap, length + 1, height + 1);
}

private static void blockify(Level level, boolean[][] blocks, int width, int height)
{
    int to = 0;
    if (levelType == LevelGenerator.TYPE_CASTLE)
        to = 4 * 2;
    else if (levelType == LevelGenerator.TYPE_UNDERGROUND)
        to = 4 * 3;

    boolean[][] b = new boolean[2][2];
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)

```

```

{
    for (int xx = x; xx <= x + 1; xx++)
    {
        for (int yy = y; yy <= y + 1; yy++)
        {
            int _xx = xx;
            int _yy = yy;
            if (_xx < 0) _xx = 0;
            if (_yy < 0) _yy = 0;
            if (_xx > width - 1) _xx = width - 1;
            if (_yy > height - 1) _yy = height - 1;
            b[xx - x][yy - y] = blocks[_xx][_yy];
        }
    }

    if (b[0][0] == b[1][0] && b[0][1] == b[1][1])
    {
        if (b[0][0] == b[0][1])
        {
            if (b[0][0])
            {
                level.setBlock(x, y, (byte) (1 + 9 * 16 + to));
            } else
            {
                // KEEP OLD BLOCK!
            }
        } else
        {
            if (b[0][0])
            {
                level.setBlock(x, y, (byte) (1 + 10 * 16 + to));
            } else
            {
                level.setBlock(x, y, (byte) (1 + 8 * 16 + to));
            }
        }
    } else if (b[0][0] == b[0][1] && b[1][0] == b[1][1])
    {
        if (b[0][0])
        {
            level.setBlock(x, y, (byte) (2 + 9 * 16 + to));
        } else
        {
            level.setBlock(x, y, (byte) (0 + 9 * 16 + to));
        }
    } else if (b[0][0] == b[1][1] && b[0][1] == b[1][0])
    {
        level.setBlock(x, y, (byte) (1 + 9 * 16 + to));
    } else if (b[0][0] == b[1][0])
    {
        if (b[0][0])
        {
            if (b[0][1])
            {
                level.setBlock(x, y, (byte) (3 + 10 * 16 + to));
            } else
            {
                level.setBlock(x, y, (byte) (3 + 11 * 16 + to));
            }
        } else
        {
            if (b[0][1])
            {
                level.setBlock(x, y, (byte) (2 + 8 * 16 + to));
            } else
            {
                level.setBlock(x, y, (byte) (0 + 8 * 16 + to));
            }
        }
    } else if (b[0][1] == b[1][1])
    {
        if (b[0][1])
        {
            if (b[0][0])
            {
                level.setBlock(x, y, (byte) (3 + 9 * 16 + to));
            } else
            {
                level.setBlock(x, y, (byte) (3 + 8 * 16 + to));
            }
        } else
        {
            if (b[0][0])
            {

```

```

        level.setBlock(x, y, (byte) (2 + 10 * 16 + to));
    } else
    {
        level.setBlock(x, y, (byte) (0 + 10 * 16 + to));
    }
}
} else
{
    level.setBlock(x, y, (byte) (0 + 1 * 16 + to));
}
}
}

//private static int buildHillStraight(int xo, int maxLength, int vfloor, final boolean isInGap)
//{
//    System.out.println("xo = " + xo);
//    int length = globalRandom.nextInt(10) + 10;
//    if (length > maxLength)
//    {
//        length = maxLength;
//    }
//    // * if( maxLength < 10 )
//    {
//        return 0;
//    }
//    // */
//    int floor = vfloor;
//    if (vfloor == DEFAULT_FLOOR)
//    {
//        floor = height - 1 - globalRandom.nextInt(4);
//    }
//    if (isInGap)
//    {
//        floor = level.height;
//    }
//    if (!isInGap)
//    {
//        for (int x = xo; x < xo + length; x++)
//        {
//            for (int y = 0; y < height; y++)
//            {
//                if (y >= floor)
//                {
//                    level.setBlock(x, y, (byte) (1 + 9 * 16));
//                }
//            }
//        }
//    }
//    addEnemiesLine(xo + 1, xo + length - 1, floor - 1);
//    int h = level.height;
//    if (isFlatLevel)
//    {
//        floor = level.height;
//    }
//    int h = isInGap ? level.height : floor;
//    int h = floor;
//    boolean keepGoing = true;
//    boolean[] occupied = new boolean[length];
//    while (keepGoing)
//    {
//        if (isFlatLevel)
//        {
//            h = vfloor;
//        }
//        else
//        {
//            h = h - 2 - globalRandom.nextInt(3);
//        }
//        if (h <= 0)
//        {
//            keepGoing = false;
//        }
//        else
//        {
//            int l = globalRandom.nextInt(maxLength) + 1;
//            int xxo = globalRandom.nextInt(length - l - 2 + 1) + xo + 1;
//            int xxo = globalRandom.nextInt(l + 1) + xo;
//            if (false) // occupied[xxo - xo] || occupied[xxo - xo + 1] || occupied[xxo - xo - 1] || occupied[xxo - xo + 1]
//            {
//                keepGoing = false;
//            }
//            else
//            {
//                occupied[xxo - xo] = true;
//                occupied[xxo - xo + 1] = true;
//                addEnemiesLine(xxo, xxo + 1, h - 1);
//            }
//        }
//    }
//}

```

```
// if (globalRandom.nextInt(4) == 0)
// {
//     decorate(xxo - 1, xxo + 1 + 1, h);
//     keepGoing = false;
// }
// for (int x = xxo; x < xxo + 1; x++)
// {
//     for (int y = h; y < floor; y++)
//     {
//         int xx = 5;
//         if (x == xxo) xx = 4;
//         if (x == xxo + 1 - 1) xx = 6;
//         int yy = 9;
//         if (y == h) yy = 8;
//
//         if (level.getBlock(x, y) == 0)
//         {
//             level.setBlock(x, y, (byte) (xx + yy * 16));
//         } else
//         {
//             if (level.getBlock(x, y) == (byte) (4 + 8 * 16))
//             level.setBlock(x, y, (byte) (4 + 11 * 16));
//             if (level.getBlock(x, y) == (byte) (6 + 8 * 16))
//             level.setBlock(x, y, (byte) (6 + 11 * 16));
//         }
//     }
// }
// }
// }
// }
// }
// }
// return length;
//}

//private static boolean canAddEnemyLine(int x0, int x1, int y)
//{
//    if (!creaturesMaskParser.canAdd())
//    {
//        return false;
//    }
//    boolean res = true;
//    for (int x = x0; x < x1; x++)
//    {
//        for (int yy = y; yy > y + 1; yy++)
//        {
//            if (level.getBlock(x, yy) != 0)
//            {
//                res = false;
//                break;
//            }
//        }
//    }
//    return res;
//}

//private static void addEnemiesLine(int x0, int x1, int y)
//{
//    if (x0 > 0)
//    return;
//    if (!canAddEnemyLine(x0, x1, y))
//    return;
//    for (int x = x0; x < x1; x++)
//    {
//        if (creaturesRandom.nextInt(25) < levelDifficulty + 1)
//        {
//            if (creaturesMaskParser.isComplete())
//            { //Difficulty of creatures on the level depends on the levelDifficulty of the level
//                int type = creaturesRandom.nextInt(4);
//                if (levelDifficulty < 1)
//                {
//                    type = CreaturesMaskParser.GOOMBA;
//                } else if (levelDifficulty < 3)
//                {
//                    int type1 = creaturesRandom.nextInt(3);
//                    int type2 = creaturesRandom.nextInt(3) + 3;
//                    type = creaturesRandom.nextInt(2) == 1 ? type1 : type2;
//                    type = creaturesMaskParser.getNativeType(type);
//                    level.setSpriteTemplate(x, y, new SpriteTemplate(type));
//                    ++counters.creatures;
//                } else

```

```

// {
// boolean enabled = false;
// int crType;// = creaturesRandom.nextInt(4);
// if (levelDifficulty < 3)
// {
// creaturesRandom.nextInt(3);
// }
// do
// {
// crType = creaturesRandom.nextInt(8);
// if (creaturesMaskParser.isEnabled(crType))
// {
// enabled = true;
// }
// }
// while (!enabled);
//
// int t = creaturesMaskParser.getNativeType(crType);
// level.setSpriteTemplate(x, y, new SpriteTemplate(t));
// ++counters.creatures;
// }
// }
// }
// }
// }

/*
first component of sum : position on Y axis
second component of sum : position on X axis
starting at 0
*16 because size of the picture is 16x16 pixels
0+9*16 -- left side of the ground
1+9*16 -- upper side of ground; common block telling "it's smth (ground) here". Is processed further.
2+9*16 -- right side of the earth
3+9*16 -- peice of the earth
9+0*16 -- block of a ladder
14+0*16 -- cannon barrel
14+1*16 -- base for cannon barrel
14+2*16 -- cannon pole
4+8*16 -- left piece of a hill of ground
4+11*16 -- left piece of a hill of ground as well
6+8*16 -- right upper peice of a hill
6+11*16 -- right upper peice of a hill on earth
2+2*16 -- animated coin
4+2+1*16 -- a rock with animated question symbol with power up
4+1+1*16 -- a rock with animated question symbol with coin
2+1*16 -- brick with power up. when broken becomes a rock
1+1*16 -- brick with power coin. when broken becomes a rock
0+1*16 -- break brick
1+10*16 -- earth, bottom piece
1+8*16 -- earth, upper piece
3+10*16 -- piece of earth
3+11*16 -- piece of earth
2+8*16 -- right part of earth
0+8*16 -- left upper part of earth
3+8*16 -- piece of earth
2+10*16 -- right bottomp iece of earth
0+10*16 -- left bottom piece of earth
*/

```

## F.3 Level Playing Module

```

package com.montywest.marioai.task

import ch.idsia.tools.MarioAIOptions

/**
 * Options that control level generation
 */
class MWLevelOptions(
    val blocks: Boolean, // Blocks appear
    val cannons: Boolean, // Cannons appear
    val coins: Boolean, // Coins appear
    val deadEnds: Boolean, // Dead ends appear in terrain forcing Mario to turn back
    val enemies: Boolean, // Enemies/Creatures appear
    val flatLevel: Boolean, // Level is flat, no change in elevation
    val frozenCreatures: Boolean, // All creatures are frozen and don't move
    val pits: Boolean, // Pits appear
    val hiddenBlocks: Boolean, // Hidden blocks appear
    val tubes: Boolean, // Tubes/Pipes appear (occasionally with piranha plant enemies
    val ladders: Boolean, // Ladders appear
    val levelDifficulty: Int, // Difficulty of level, effective range 0-25, 0 easiest

```



```

    val levelLength: Int, // Length of level in blocks
    val levelType: Int, // Type of level, 0-Outside 1-Cave, 2-Castle
    val startingMarioMode: Int, // Mode Mario starts as 0-small, 1-big, 2-fire
    val timeLimit: Int // Number of Mario seconds allowed to complete level
  ) {

    override def clone: MWLevelOptions = {
      new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel, this.
        frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withBlocks(blocks: Boolean): MWLevelOptions = {
      return new MWLevelOptions(blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel, this.
        frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withCannons(cannons: Boolean): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel, this.
        frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withCoins(coins: Boolean): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, this.cannons, coins, this.deadEnds, this.enemies, this.flatLevel, this.
        frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withDeadEnds(deadEnds: Boolean): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, this.cannons, this.coins, deadEnds, this.enemies, this.flatLevel, this.
        frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withEnemies(enemies: Boolean): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, enemies, this.flatLevel, this.
        frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withFlatLevel(flatLevel: Boolean): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, flatLevel, this.
        frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withFrozenCreatures(frozenCreatures: Boolean): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel,
        frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withPits(pits: Boolean): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel,
        this.frozenCreatures, pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withHiddenBlocks(hiddenBlocks: Boolean): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel,
        this.frozenCreatures, this.pits, hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withTubes(tubes: Boolean): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel,
        this.frozenCreatures, this.pits, this.hiddenBlocks, tubes, this.ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withLadders(ladders: Boolean): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel,
        this.frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, ladders, this.levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }

    def withLevelDifficulty(levelDifficulty: Int): MWLevelOptions = {
      return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel,
        this.frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, levelDifficulty, this.
        levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
    }
  }

```

```

def withLevelLength(_levelLength: Int): MWLevelOptions = {
  return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel,
    this.frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty,
    _levelLength, this.levelType, this.startingMarioMode, this.timeLimit)
}

def withLevelType(levelType: Int): MWLevelOptions = {
  return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel,
    this.frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty,
    this.levelLength, levelType, this.startingMarioMode, this.timeLimit)
}

def withStartingMarioMode(startingMarioMode: Int): MWLevelOptions = {
  return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel,
    this.frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty,
    this.levelLength, this.levelType, startingMarioMode, this.timeLimit)
}

def withTimeLimit(_timeLimit: Int): MWLevelOptions = {
  return new MWLevelOptions(this.blocks, this.cannons, this.coins, this.deadEnds, this.enemies, this.flatLevel,
    this.frozenCreatures, this.pits, this.hiddenBlocks, this.tubes, this.ladders, this.levelDifficulty,
    this.levelLength, this.levelType, this.startingMarioMode, _timeLimit)
}

override def toString: String = {
  "Level Options :- \n" +
  " blocks: " + blocks + "\n" +
  " cannons: " + cannons + "\n" +
  " coins: " + coins + "\n" +
  " deadEnds: " + deadEnds + "\n" +
  " enemies: " + enemies + "\n" +
  " flatLevel: " + flatLevel + "\n" +
  " frozenCreatures: " + frozenCreatures + "\n" +
  " gaps: " + pits + "\n" +
  " hiddenBlocks: " + hiddenBlocks + "\n" +
  " tubes: " + tubes + "\n" +
  " ladders: " + ladders + "\n" +
  " levelDifficulty: " + levelDifficulty + "\n" +
  " levelLength: " + levelLength + "\n" +
  " levelType: " + levelType + "\n" +
  " startingMarioMode: " + startingMarioMode + "\n" +
  " timeLimit: " + timeLimit + "\n"
}

override def equals(obj: Any): Boolean = {
  obj match {
    case other: MWLevelOptions => {
      if (other == null) false
      (blocks == other.blocks &&
        cannons == other.cannons &&
        coins == other.coins &&
        deadEnds == other.deadEnds &&
        enemies == other.enemies &&
        flatLevel == other.flatLevel &&
        frozenCreatures == other.frozenCreatures &&
        pits == other.pits &&
        hiddenBlocks == other.hiddenBlocks &&
        tubes == other.tubes &&
        ladders == other.ladders &&
        levelDifficulty == other.levelDifficulty &&
        levelLength == other.levelLength &&
        levelType == other.levelType &&
        startingMarioMode == other.startingMarioMode &&
        timeLimit == other.timeLimit)
    }
    case _ => false
  }
}

object MWLevelOptions {

  def updateMarioAIOptions(marioAIOptions: MarioAIOptions, levelOptions: MWLevelOptions): MarioAIOptions = {
    marioAIOptions.setBlocksCount(levelOptions.blocks)
    marioAIOptions.setCannonsCount(levelOptions.cannons)
    marioAIOptions.setCoinsCount(levelOptions.coins)
    marioAIOptions.setDeadEndsCount(levelOptions.deadEnds)
    marioAIOptions.setEnemies(if(levelOptions.enemies) "" else "off")
    marioAIOptions.setFlatLevel(levelOptions.flatLevel)
    marioAIOptions.setFrozenCreatures(levelOptions.frozenCreatures)
    marioAIOptions.setGapsCount(levelOptions.pits)
    marioAIOptions.setHiddenBlocksCount(levelOptions.hiddenBlocks)
    marioAIOptions.setTubesCount(levelOptions.tubes)
  }
}

```

```

        marioAIOptions.setLevelLadder(levelOptions.ladders)
        marioAIOptions.setLevelDifficulty(levelOptions.levelDifficulty)
        marioAIOptions.setLevelLength(levelOptions.levelLength)
        marioAIOptions.setLevelType(levelOptions.levelType)
        marioAIOptions.setMarioMode(levelOptions.startingMarioMode)
        marioAIOptions.setTimeLimit(levelOptions.timeLimit)

    marioAIOptions
}

val defaultOptions: MWLevelOptions =
    new MWLevelOptions(true, //blocks
        true, //cannons
        true, //coins
        false, //deadEnds
        true, //enemies
        false, //flatLevel
        false, //frozenCreatures
        true, //gaps
        false, //hiddenBlocks
        false, //tubes
        false, //ladders
        0, //levelDifficulty
        256, //levelLength
        0, //levelType
        2, //startingMarioMode
        200 //timeLimit
    )

val noUpdate: (Int, MWLevelOptions) => MWLevelOptions = (i: Int, options: MWLevelOptions) => options

val compNumberOfLevels = 512;
val compOptions = defaultOptions
def compUpdate(levelSeed: Int): (Int, MWLevelOptions) => MWLevelOptions = (i: Int, options: MWLevelOptions) =>
{
    // println("Prev ll; " + options.levelLength)
    options.withLevelLength((((i+levelSeed) * 431) % (501+levelSeed) ) % 462) + 50)
        .withTimeLimit((options.levelLength * 0.7).toInt)
        .withLevelType(i % 3)
        .withLevelDifficulty((compNumberOfLevels - i)/32)
        .withPits(i % 4 != 2)
        .withCannons(i % 6 == 2)
        .withTubes(i % 5 == 1)
        .withCoins(i % 5 != 0)
        .withBlocks(i % 6 != 2)
    // .withHiddenBlocks(i % 6 != 0)
        .withDeadEnds(false)
        .withLadders(i % 10 == 2)
        .withFrozenCreatures(i % 3 == 1)
        .withEnemies(!(i % 4 == 1))
        .withStartingMarioMode(
            if (i % 7 == 5 || i % 7 == 1) {
                if (i % 2 == 0) 0
                else 1
            } else
                2)
}
}

```

---

```

package com.montywest.marioai.task;

import ch.idsia.benchmark.tasks.SystemOfValues;

public class SystemOfValuesAdapter extends SystemOfValues {

    public SystemOfValuesAdapter(
        int distance,
        int win,
        int mode,
        int coins,
        int flowerFire,
        int mushroom,
        int kills,
        int killedByFire,
        int killedByShell,
        int killedByStomp,
        int timeLeft,
        int hiddenBlock,
        int greenMushroom,
        int stomp
    ) {
        this.distance = distance;
    }
}

```

```

        this.win = win;
        this.mode = mode;
        this.coins = coins;
        this.flowerFire = flowerFire;
        this.mushroom = mushroom;
        this.kills = kills;
        this.killedByFire = killedByFire;
        this.killedByShell = killedByShell;
        this.killedByStomp = killedByStomp;
        this.timeLeft = timeLeft;
        this.hiddenBlock = hiddenBlock;
        this.greenMushroom = greenMushroom;
        this.stomp = stomp;
    }
}

```

---

```

package com.montywest.marioai.task

import scala.language.implicitConversions
import ch.idisia.benchmark.tasks.SystemOfValues

/**
 * Multipliers for several level playing
 * statistics.
 * Comments describe the statistic.
 * For example, if Mario completes the level
 * It will be win * 1, otherwise it will win * 0
 */
class MWEvaluationMultipliers(
    val distance: Int, // Distance travelled by Mario in pixels (16 pixels to a block)
    val win: Int, // 1 for level complete, 0 otherwise
    val mode: Int, // Mario's final mode on completion or death, 2-fire, 1-big, 0-small
    val coins: Int, // Number of coins collected
    val flowerFire: Int, // Number fire flowers collected
    val kills: Int, // Number of enemy kills
    val killedByFire: Int, // Number of kills by fireball
    val killedByShell: Int, // Number of kills by shell
    val killedByStomp: Int, // Number of kills by stomp
    val mushroom: Int, // Number of mushrooms collected
    val timeLeft: Int, // Mario seconds left on completion, 0 if level not completed
    val hiddenBlock: Int, // Number of hidden blocks hit
    val greenMushroom: Int, // Number of green mushrooms
    val stomp: Int) // Unused
{

    override def clone: MWEvaluationMultipliers = {
        new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
            killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
    }

    def withDistance(distance: Int): MWEvaluationMultipliers = {
        new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
            killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
    }

    def withWin(win: Int): MWEvaluationMultipliers = {
        new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
            killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
    }

    def withMode(mode: Int): MWEvaluationMultipliers = {
        new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
            killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
    }

    def withCoins(coins: Int): MWEvaluationMultipliers = {
        new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
            killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
    }

    def withFlowerFire(flowerFire: Int): MWEvaluationMultipliers = {
        new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
            killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
    }

    def withKills(kills: Int): MWEvaluationMultipliers = {
        new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
            killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
    }

    def withKilledByFire(killedByFire: Int): MWEvaluationMultipliers = {

```

```

    new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
        killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
}

def withKilledByShell(killedByShell: Int): MWEvaluationMultipliers = {
    new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
        killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
}

def withKilledByStomp(killedByStomp: Int): MWEvaluationMultipliers = {
    new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
        killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
}

def withMushroom(mushroom: Int): MWEvaluationMultipliers = {
    new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
        killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
}

def withTimeLeft(timeLeft: Int): MWEvaluationMultipliers = {
    new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
        killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
}

def withHiddenBlock(hiddenBlock: Int): MWEvaluationMultipliers = {
    new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
        killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
}

def withGreenMushroom(greenMushroom: Int): MWEvaluationMultipliers = {
    new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
        killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
}

def withStomp(stomp: Int): MWEvaluationMultipliers = {
    new MWEvaluationMultipliers(distance, win, mode, coins, flowerFire, kills, killedByFire, killedByShell,
        killedByStomp, mushroom, timeLeft, hiddenBlock, greenMushroom, stomp)
}

override def toString: String = {
    "Evaluation Multipliers :-\n" +
    " distance: " + distance + "\n" +
    " win: " + win + "\n" +
    " mode: " + mode + "\n" +
    " coins: " + coins + "\n" +
    " flowerFire: " + flowerFire + "\n" +
    " kills: " + kills + "\n" +
    " killedByFire: " + killedByFire + "\n" +
    " killedByShell: " + killedByShell + "\n" +
    " killedByStomp: " + killedByStomp + "\n" +
    " mushroom: " + mushroom + "\n" +
    " timeLeft: " + timeLeft + "\n" +
    " hiddenBlock: " + hiddenBlock + "\n" +
    " greenMushroom: " + greenMushroom + "\n" +
    " stomp: " + stomp + "\n"
}

override def equals(obj: Any): Boolean = {
    obj match {
        case other: MWEvaluationMultipliers => {
            if (other == null) false
            (distance == other.distance &&
            win == other.win &&
            mode == other.mode &&
            coins == other.coins &&
            flowerFire == other.flowerFire &&
            mushroom == other.mushroom &&
            kills == other.kills &&
            killedByFire == other.killedByFire &&
            killedByShell == other.killedByShell &&
            killedByStomp == other.killedByStomp &&
            timeLeft == other.timeLeft &&
            hiddenBlock == other.hiddenBlock &&
            greenMushroom == other.greenMushroom &&
            stomp == other.stomp)
        }
        case _ => false
    }
}

object MWEvaluationMultipliers {
    implicit def toAdapter(values: MWEvaluationMultipliers): SystemOfValuesAdapter = {
        new SystemOfValuesAdapter(

```

```

        values.distance,
        values.win,
        values.mode,
        values.coins,
        values.flowerFire,
        values.mushroom,
        values.kills,
        values.killedByFire,
        values.killedByShell,
        values.killedByStomp,
        values.timeLeft,
        values.hiddenBlock,
        values.greenMushroom,
        values.stomp)
    }

    val defaultEvaluationMultipliers =
    new MWEvaluationMultipliers(
        1, //Distance
        2048, //Win
        16, //Mode
        16, //Coins
        64, //FlowerFire
        58, //Mushroom
        42, //Kills
        4, //KilledByFire
        17, //KilledByShell
        12, //KilledByStomp
        8, //TimeLeft
        24, //HiddenBlock
        58, //GreenMushroom
        10 //Stomp
    )

    val compEvaluationMultipliers = defaultEvaluationMultipliers

    val zeroEvaluationMultipliers =
    new MWEvaluationMultipliers(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
}

```

---

```

package com.montywest.marioai.task

import ch.idsia.benchmark.tasks.BasicTask
import ch.idsia.benchmark.tasks.Task
import ch.idsia.tools.MarioAIOptions
import ch.idsia.tools.EvaluationInfo
import java.text.DecimalFormat
import scala.annotation.tailrec
import ch.idsia.agents.Agent

abstract class MWBasicTask(val name: String,
    val baseLevelOptions: MWLevelOptions,
    val updateOptionsFunc: (Int, MWLevelOptions) => MWLevelOptions,
    val visualisation: Boolean,
    val args: Array[String],
    val saveLevelScores: Boolean)
    extends BasicTask({
        val marioAIOptions = new MarioAIOptions(args)
        marioAIOptions.setVisualization(visualisation)
        marioAIOptions
    }) with Task {

    var levelScores: Seq[Int] = Nil
    val localEvaluationInfo = new EvaluationInfo();
    var disqualifications = 0;
    localEvaluationInfo.setTaskName(name);
    this.resetLocalEvaluationInfo

    def getFitness: Int = localEvaluationInfo.computeWeightedFitness()

    def numberOfLevels: Int

    protected def updateOptions(episode: Int, options: MWLevelOptions): MWLevelOptions = updateOptionsFunc(episode,
        options)

    protected def nextLevelSeed(episode: Int, lastSeed: Int) = {
        (3*episode) + lastSeed
    }

    protected def getBaseLevelSeed: Int

    protected def getLevelScore(envEvalInfo: EvaluationInfo): Int

```

```

override def doEpisodes(amount: Int, verbose: Boolean, repetitionsOfSingleEpisode: Int): Unit = {
  @tailrec
  def runSingle(iteration: Int, prevOptions: MWLevelOptions, disqualifications: Int): Int = {
    if (iteration == amount) {
      finishLevelScores
      disqualifications
    } else {
      val newOptions = updateOptions(iteration, prevOptions)

      val marioAIOptions = MWLevelOptions.updateMarioAIOptions(options, newOptions)
      marioAIOptions.setLevelRandSeed(nextLevelSeed(iteration, marioAIOptions.getLevelRandSeed))

      super.setOptionsAndReset(marioAIOptions)
      val disqualified: Int = if (!runSingleEpisode(repetitionsOfSingleEpisode)) 1 else 0

      logLevelScore(iteration, getLevelScore(super.getEvaluationInfo))
      updateLocalEvaluationInfo(super.getEvaluationInfo)
      runSingle(iteration+1, newOptions, disqualifications + disqualified)
    }
  }

  super.setOptionsAndReset(MWLevelOptions.updateMarioAIOptions(options, baseLevelOptions))
  disqualifications = runSingle(0, baseLevelOptions, 0)
}

def doEpisodes: Unit = {
  doEpisodes(numberOfLevels, false, 1)
}

protected def injectAgent(agent: Agent, resetEval: Boolean): Unit = {
  options.setAgent(agent);
  reset
  if (resetEval) this.resetLocalEvaluationInfo
}

protected def injectLevelSeed(seed: Int, resetEval: Boolean): Unit = {
  options.setLevelRandSeed(seed)
  reset
  if (resetEval) this.resetLocalEvaluationInfo
}

private def logLevelScore(ep: Int, score: Int): Unit = {
  if (visualisation) {
    println("Episode " + ep + " score: " + score)
  }
  if (saveLevelScores) {
    levelScores = score +: levelScores
  }
}

private def finishLevelScores: Unit = {
  if (saveLevelScores) {
    levelScores = levelScores.reverse
  }
}

def getStatistics(): String = {
  val df: DecimalFormat = new DecimalFormat("#.##")
  return "\n[MarioAI] ~ Evaluation Results for Task - " + localEvaluationInfo.getTaskName() + "\n" +
    "\n Seed : " + this.getBaseLevelSeed +
    "\n Agent : " + options.getAgent.getName +
    "\n Weighted Fitness : " + getFitness +
    "\n Levels Completed : " + localEvaluationInfo.marioStatus + " of " + numberOfLevels +
    "\n Mario Mode : " + localEvaluationInfo.marioMode +
    "\n Collisions with creatures : " + localEvaluationInfo.collisionsWithCreatures +
    "\n Passed (Cells, Phys) : " + localEvaluationInfo.distancePassedCells + " of " + localEvaluationInfo.
      levelLength + ", " + df.format(localEvaluationInfo.distancePassedPhys) + " of " + df.format(
        localEvaluationInfo.levelLength * 16) + " (" + localEvaluationInfo.distancePassedCells * 100 /
        localEvaluationInfo.levelLength + "% passed)" +
    "\n Time Spent(marioseconds) : " + localEvaluationInfo.timeSpent +
    "\n Time Left(marioseconds) : " + localEvaluationInfo.timeLeft +
    "\n Coins Gained : " + localEvaluationInfo.coinsGained + " of " + localEvaluationInfo.
      totalNumberOfCoins + " (" + localEvaluationInfo.coinsGained * 100 / (if(localEvaluationInfo.
        totalNumberOfCoins == 0) 1 else localEvaluationInfo.totalNumberOfCoins) + "% collected)" +
    "\n Hidden Blocks Found : " + localEvaluationInfo.hiddenBlocksFound + " of " + localEvaluationInfo.
      totalNumberOfHiddenBlocks + " (" + localEvaluationInfo.hiddenBlocksFound * 100 / (if(
        localEvaluationInfo.totalNumberOfHiddenBlocks == 0) 1 else localEvaluationInfo.
        totalNumberOfHiddenBlocks) + "% found)" +
    "\n Mushrooms Devoured : " + localEvaluationInfo.mushroomsDevoured + " of " + localEvaluationInfo.
      totalNumberOfMushrooms + " found (" + localEvaluationInfo.mushroomsDevoured * 100 / (if(
        localEvaluationInfo.totalNumberOfMushrooms == 0) 1 else localEvaluationInfo.
        totalNumberOfMushrooms) + "% collected)" +

```

```

        "\n Flowers Devoured : " + localEvaluationInfo.flowersDevoured + " of " + localEvaluationInfo.
            totalNumberOfFlowers + " found (" + localEvaluationInfo.flowersDevoured * 100 / (if(
                localEvaluationInfo.totalNumberOfFlowers == 0) 1 else localEvaluationInfo.totalNumberOfFlowers)
            + "% collected)" +
        "\n Kills Total : " + localEvaluationInfo.killsTotal + " of " + localEvaluationInfo.
            totalNumberOfCreatures + " found (" + localEvaluationInfo.killsTotal * 100 / (if(
                localEvaluationInfo.totalNumberOfCreatures == 0) 1 else localEvaluationInfo.
                totalNumberOfCreatures) + "%)" +
        "\n Kills By Fire : " + localEvaluationInfo.killsByFire +
        "\n Kills By Shell : " + localEvaluationInfo.killsByShell +
        "\n Kills By Stomp : " + localEvaluationInfo.killsByStomp +
        "\n Disqualifications : " + disqualifications
    }

    protected def updateLocalEvaluationInfo(evInfo: EvaluationInfo) = {
        try {
            localEvaluationInfo.distancePassedCells += evInfo.distancePassedCells;
            localEvaluationInfo.distancePassedPhys += evInfo.distancePassedPhys;
            localEvaluationInfo.flowersDevoured += evInfo.flowersDevoured;
            localEvaluationInfo.killsTotal += evInfo.killsTotal;
            localEvaluationInfo.killsByFire += evInfo.killsByFire;
            localEvaluationInfo.killsByShell += evInfo.killsByShell;
            localEvaluationInfo.killsByStomp += evInfo.killsByStomp;
            localEvaluationInfo.marioMode += evInfo.marioMode;
            localEvaluationInfo.marioStatus += evInfo.marioStatus;
            localEvaluationInfo.mushroomsDevoured += evInfo.mushroomsDevoured;
            localEvaluationInfo.coinsGained += evInfo.coinsGained;
            localEvaluationInfo.timeLeft += evInfo.timeLeft;
            localEvaluationInfo.timeSpent += evInfo.timeSpent;
            localEvaluationInfo.hiddenBlocksFound += evInfo.hiddenBlocksFound;
            localEvaluationInfo.totalNumberOfCoins += evInfo.totalNumberOfCoins;
            localEvaluationInfo.totalNumberOfCreatures += evInfo.totalNumberOfCreatures;
            localEvaluationInfo.totalNumberOfFlowers += evInfo.totalNumberOfFlowers;
            localEvaluationInfo.totalNumberOfMushrooms += evInfo.totalNumberOfMushrooms;
            localEvaluationInfo.totalNumberOfHiddenBlocks += evInfo.totalNumberOfHiddenBlocks;
            localEvaluationInfo.collisionsWithCreatures += evInfo.collisionsWithCreatures;
            localEvaluationInfo.levelLength += evInfo.levelLength;
        } catch {
            case e: NullPointerException => throw new RuntimeException("Strikes again! lei: " + (localEvaluationInfo !=
                null) + " eei: " + (evInfo != null), e)
        }
    }

    protected def resetLocalEvaluationInfo = {
        localEvaluationInfo.distancePassedCells = 0;
        localEvaluationInfo.distancePassedPhys = 0;
        localEvaluationInfo.flowersDevoured = 0;
        localEvaluationInfo.killsTotal = 0;
        localEvaluationInfo.killsByFire = 0;
        localEvaluationInfo.killsByShell = 0;
        localEvaluationInfo.killsByStomp = 0;
        localEvaluationInfo.marioMode = 0;
        localEvaluationInfo.marioStatus = 0;
        localEvaluationInfo.mushroomsDevoured = 0;
        localEvaluationInfo.coinsGained = 0;
        localEvaluationInfo.timeLeft = 0;
        localEvaluationInfo.timeSpent = 0;
        localEvaluationInfo.hiddenBlocksFound = 0;
        localEvaluationInfo.totalNumberOfCoins = 0;
        localEvaluationInfo.totalNumberOfCreatures = 0;
        localEvaluationInfo.totalNumberOfFlowers = 0;
        localEvaluationInfo.totalNumberOfMushrooms = 0;
        localEvaluationInfo.totalNumberOfHiddenBlocks = 0;
        localEvaluationInfo.collisionsWithCreatures = 0;
        localEvaluationInfo.levelLength = 0;
    }
}

}

package com.montywest.marioai.task

import ch.idsia.agents.Agent

trait EvaluationTask {

    def evaluate: Int;

    def withAgent(agent: Agent): EvaluationTask;

    def withLevelSeed(seed: Int): EvaluationTask;

    def getLevelSeed: Int;

```



```

    def getStatistics: String;
}

package com.montywest.marioai.task

import ch.idsia.tools.MarioAIOptions
import ch.idsia.agents.Agent
import ch.idsia.tools.EvaluationInfo

class MWEvaluationTask(val numberOfLevels: Int,
    val evalValues: MWEvaluationMultipliers,
    override val baseLevelOptions: MWLevelOptions,
    override val updateOptionsFunc: (Int, MWLevelOptions) => MWLevelOptions,
    override val visualisation: Boolean,
    override val args: Array[String],
    override val saveLevelScores: Boolean)
    extends MWBasicTask("MWMainPlayTask", baseLevelOptions, updateOptionsFunc, visualisation,
        args, saveLevelScores) with EvaluationTask {

    private var baseLevelSeed: Int = 0;

    override def nextLevelSeed(episode: Int, lastSeed: Int) = {
        (3*episode) + lastSeed
    }

    override def getFitness: Int = {
        localEvaluationInfo.computeWeightedFitness(evalValues)
    }

    override def evaluate: Int = {
        doEpisodes
        getFitness
    }

    override def withAgent(agent: Agent): MWEvaluationTask = {
        super.injectAgent(agent, true)
        this
    }

    override def withLevelSeed(seed: Int): MWEvaluationTask = {
        baseLevelSeed = seed
        super.injectLevelSeed(seed, true)
        this
    }

    override def getLevelSeed: Int = {
        options.getLevelRandSeed
    }

    override def getBaseLevelSeed: Int = {
        baseLevelSeed
    }

    override def getLevelScore(eval: EvaluationInfo): Int = {
        eval.computeWeightedFitness(evalValues)
    }
}

object MWEvaluationTask {

    def apply(numberOfLevels: Int,
        evalValues: MWEvaluationMultipliers,
        baseLevelOptions: MWLevelOptions,
        updateOptionsFunc: (Int, MWLevelOptions) => MWLevelOptions,
        visualisation: Boolean = true,
        args: Array[String] = Array.empty,
        saveLevelScores: Boolean = false): MWEvaluationTask = {

        new MWEvaluationTask(numberOfLevels, evalValues, baseLevelOptions, updateOptionsFunc, visualisation, args,
            saveLevelScores)
    }
}

package com.montywest.marioai.learning.ec.params

import com.montywest.marioai.task.MWLevelOptions

```

```

import ec.util.ParameterDatabaseInf
import ec.util.Parameter
import java.util.regex.Pattern
import com.montywest.marioai.task.MWEvaluationMultipliers

object EvaluationParamsUtil {

  val P_EVAL_BASE = "eval.problem"

  val P_NUM_OF_LEVELS = "num-levels"

  //LEVEL OPTIONS
  val P_LEVEL_BASE = "level"

  val P_BASE_OPTIONS = "base"

  val P_BLOCKS = "blocks"
  val P_CANNONS = "cannons"
  val P_COINS = "coins"
  val P_DEAD_ENDS = "dead-ends"
  val P_ENEMIES = "enemies"
  val P_FLAT = "flat"
  val P_FROZEN_CREATURES = "frozen-enemies"
  val P_PITS = "pits"
  val P_HIDDEN_BLOCKS = "hidden-blocks"
  val P_TUBES = "tubes"
  val P_DIFFICULTY = "difficulty-num"
  val P_LENGTH = "length-num"
  val P_TYPE = "type-num"
  val P_START_MODE = "start-mode-num"
  val P_TIME_LIMIT = "time-limit"

  val blocksFn = (b: Boolean, opt: MWLevelOptions) => opt.withBlocks(b)
  val cannonsFn = (b: Boolean, opt: MWLevelOptions) => opt.withCannons(b)
  val coinsFn = (b: Boolean, opt: MWLevelOptions) => opt.withCoins(b)
  val deadEndsFn = (b: Boolean, opt: MWLevelOptions) => opt.withDeadEnds(b)
  val enemiesFn = (b: Boolean, opt: MWLevelOptions) => opt.withEnemies(b)
  val flatLevelFn = (b: Boolean, opt: MWLevelOptions) => opt.withFlatLevel(b)
  val frozenFn = (b: Boolean, opt: MWLevelOptions) => opt.withFrozenCreatures(b)
  val pitsFn = (b: Boolean, opt: MWLevelOptions) => opt.withPits(b)
  val hiddenBlocksFn = (b: Boolean, opt: MWLevelOptions) => opt.withHiddenBlocks(b)
  val tubesFn = (b: Boolean, opt: MWLevelOptions) => opt.withTubes(b)

  val difficultyFn = (i: Int, opt: MWLevelOptions) => opt.withLevelDifficulty(i)
  val lengthFn = (i: Int, opt: MWLevelOptions) => opt.withLevelLength(i)
  val typeFn = (i: Int, opt: MWLevelOptions) => opt.withLevelType(i)
  val startModeFn = (i: Int, opt: MWLevelOptions) => opt.withStartingMarioMode(i)
  val timeLimitFn = (i: Int, opt: MWLevelOptions) => opt.withTimeLimit(i)

  //EVAL MULTIPLIER

  val P_MULTS_BASE = "mults"

  val P_DISTANCE = "distance"
  val P_WIN = "win"
  val P_MODE = "mode"
  val P_FLOWER_FIRE = "flower-fire"
  val P_KILLS = "kills"
  val P_KILLED_BY_FIRE = "kills-fire"
  val P_KILLED_BY_SHELL = "kills-shell"
  val P_KILLED_BY_STOMP = "kills-stomp"
  val P_MUSHROOM = "mushroom"
  val P_TIME_LEFT = "time-left"
  val P_HIDDEN_BLOCK = "hidden-block"
  val P_GREEN_MUSHROOM = "green-mushroom"
  val P_STOMP = "stomp"

  //SEEDS
  val P_SEED = "seed"
  val P_SEED_START = "start"
  val P_SEED_ADD = "add"
  val P_SEED_MULT = "mult"

  def getLevelParamsBase(pd: ParameterDatabaseInf): Option[Parameter] = {
    getParamsBase(pd, P_LEVEL_BASE)
  }

  def getMultsParamsBase(pd: ParameterDatabaseInf): Option[Parameter] = {
    getParamsBase(pd, P_MULTS_BASE)
  }
}

```

```

def getSeedParamBase(pd: ParameterDatabaseInf): Option[Parameter] = {
  val bp = getParamsBase(pd, P_SEED)
  bp
}

private def getParamsBase(pd: ParameterDatabaseInf, postfix: String): Option[Parameter] = {
  val evalBase = new Parameter("eval").push("problem").push(postfix)
  if (pd.exists(evalBase, null))
    Some(evalBase)
  else {
    val probBase = new Parameter("problem").push(postfix)
    if (pd.exists(probBase, null))
      Some(probBase)
    else {
      val plainBase = new Parameter(postfix)
      if (pd.exists(plainBase, null))
        Some(plainBase)
      else
        None
    }
  }
}

def getLevelSeeds(pd: ParameterDatabaseInf, base: Parameter, dStart: Int, dAdd: Int, dMult: Int): (Int, Int, Int) = {
  val seedStart = pd.getIntWithDefault(base.push(P_SEED_START), null, dStart)
  val seedAdd = pd.getIntWithDefault(base.push(P_SEED_ADD), null, dAdd)
  val seedMult = pd.getIntWithDefault(base.push(P_SEED_MULT), null, dMult)
  (seedStart, seedAdd, seedMult)
}

def getNumberOfLevels(pd: ParameterDatabaseInf, base: Parameter): Option[Int] = {
  if (pd.exists(base.push(P_NUM_OF_LEVELS), null))
    Some(pd.getInt(base.push(P_NUM_OF_LEVELS), null, 1))
  else
    None
}

def getEvaluationMultipliers(pd: ParameterDatabaseInf, base: Parameter): MWEvaluationMultipliers = {
  def innerLoop(opt: MWEvaluationMultipliers, vec: Seq[(String, (Int, MWEvaluationMultipliers) => MWEvaluationMultipliers)]): MWEvaluationMultipliers = vec match {
    case Nil => opt
    case (key: String, fn: ((Int, MWEvaluationMultipliers) => MWEvaluationMultipliers)) +: tl => {
      innerLoop(getIntOption(pd, base, key) match { case Some(i) => {fn(i, opt)} case None => opt }, tl)
    }
  }

  innerLoop(
    MWEvaluationMultipliers.zeroEvaluationMultipliers,
    Seq((P_DISTANCE, (i: Int, eval: MWEvaluationMultipliers) => eval.withDistance(i)),
      (P_WIN, (i: Int, eval: MWEvaluationMultipliers) => eval.withWin(i)),
      (P_MODE, (i: Int, eval: MWEvaluationMultipliers) => eval.withMode(i)),
      (P_FLOWER_FIRE, (i: Int, eval: MWEvaluationMultipliers) => eval.withFlowerFire(i)),
      (P_COINS, (i: Int, eval: MWEvaluationMultipliers) => eval.withCoins(i)),
      (P_KILLS, (i: Int, eval: MWEvaluationMultipliers) => eval.withKills(i)),
      (P_KILLED_BY_FIRE, (i: Int, eval: MWEvaluationMultipliers) => eval.withKilledByFire(i)),
      (P_KILLED_BY_SHELL, (i: Int, eval: MWEvaluationMultipliers) => eval.withKilledByShell(i)),
      (P_KILLED_BY_STOMP, (i: Int, eval: MWEvaluationMultipliers) => eval.withKilledByStomp(i)),
      (P_MUSHROOM, (i: Int, eval: MWEvaluationMultipliers) => eval.withMushroom(i)),
      (P_TIME_LEFT, (i: Int, eval: MWEvaluationMultipliers) => eval.withTimeLeft(i)),
      (P_HIDDEN_BLOCK, (i: Int, eval: MWEvaluationMultipliers) => eval.withHiddenBlock(i)),
      (P_GREEN_MUSHROOM, (i: Int, eval: MWEvaluationMultipliers) => eval.withGreenMushroom(i)),
      (P_STOMP, (i: Int, eval: MWEvaluationMultipliers) => eval.withStomp(i)))
  )
}

def getBaseLevelOptions(pd: ParameterDatabaseInf, preBase: Parameter): MWLevelOptions = {
  val base = preBase.push(P_BASE_OPTIONS);

  def innerBoolLoop(opt: MWLevelOptions, vec: Seq[(String, (Boolean, MWLevelOptions) => MWLevelOptions)]): MWLevelOptions = vec match {
    case Nil => opt
    case (key: String, fn: ((Boolean, MWLevelOptions) => MWLevelOptions)) +: tl => {
      innerBoolLoop(getBooleanOption(pd, base, key) match { case Some(b) => {fn(b, opt)} case None => opt }, tl)
    }
  }

  def innerIntLoop(opt: MWLevelOptions, vec: Seq[(String, (Int, MWLevelOptions) => MWLevelOptions)]): MWLevelOptions = vec match {
    case Nil => opt
    case (key: String, fn: ((Int, MWLevelOptions) => MWLevelOptions)) +: tl => {
      innerIntLoop(getIntOption(pd, base, key) match { case Some(i) => {fn(i, opt)} case None => opt }, tl)
    }
  }
}

```

```

val afterBools =
  innerBoolLoop(
    MWLevelOptions.defaultOptions,
    Seq((P_BLOCKS, blocksFn),
        (P_CANNONS, cannonsFn),
        (P_COINS, coinsFn),
        (P_DEAD_ENDS, deadEndsFn),
        (P_ENEMIES, enemiesFn),
        (P_FLAT, flatLevelFn),
        (P_FROZEN_CREATURES, frozenFn),
        (P_PITS, pitsFn),
        (P_HIDDEN_BLOCKS, hiddenBlocksFn),
        (P_TUBES, tubesFn))
  )

  innerIntLoop(
    afterBools,
    Seq((P_DIFFICULTY, difficultyFn),
        (P_LENGTH, lengthFn),
        (P_TYPE, typeFn),
        (P_START_MODE, startModeFn),
        (P_TIME_LIMIT, timeLimitFn))
  )
}

def getUpdateLevelOptionsFunction(pd: ParameterDatabaseInf, base: Parameter, numberOfLevels: Int): (Int,
  MWLevelOptions) => MWLevelOptions = {

  def innerBool(mapOpt: Option[Map[Int, Boolean]], key: String, i: Int) = {
    getBooleanOption(pd, base.push(" "+i), key) match {
      case None => mapOpt
      case Some(b: Boolean) => mapOpt match {
        case None => Some(Map(i->b))
        case Some(map) => Some(map + (i->b))
      }
    }
  }

  def innerInt(mapOpt: Option[Map[Int, Int]], key: String, i: Int) = {
    getIntOption(pd, base.push(" "+i), key) match {
      case None => mapOpt
      case Some(n: Int) => mapOpt match {
        case None => Some(Map(i->n))
        case Some(map) => Some(map + (i->n))
      }
    }
  }

  var blocks: Option[Map[Int, Boolean]] = None
  var cannons: Option[Map[Int, Boolean]] = None
  var coins: Option[Map[Int, Boolean]] = None
  var deadEnds: Option[Map[Int, Boolean]] = None
  var enemies: Option[Map[Int, Boolean]] = None
  var flatLevel: Option[Map[Int, Boolean]] = None
  var frozenCreatues: Option[Map[Int, Boolean]] = None
  var pits: Option[Map[Int, Boolean]] = None
  var hiddenBlocks: Option[Map[Int, Boolean]] = None
  var tubes: Option[Map[Int, Boolean]] = None
  var levelDifficulty: Option[Map[Int, Int]] = None
  var levelLength: Option[Map[Int, Int]] = None
  var levelType: Option[Map[Int, Int]] = None
  var startMode: Option[Map[Int, Int]] = None
  var timeLimit: Option[Map[Int, Int]] = None

  for (i <- 0 until numberOfLevels) {
    // if (pd.exists(base.push(" "+i), null)) {
      blocks = innerBool(blocks, P_BLOCKS, i)
      cannons = innerBool(cannons, P_CANNONS, i)
      coins = innerBool(coins, P_COINS, i)
      deadEnds = innerBool(deadEnds, P_DEAD_ENDS, i)
      enemies = innerBool(enemies, P_ENEMIES, i)
      flatLevel = innerBool(flatLevel, P_FLAT, i)
      frozenCreatues = innerBool(frozenCreatues, P_FROZEN_CREATURES, i)
      pits = innerBool(pits, P_PITS, i)
      hiddenBlocks = innerBool(hiddenBlocks, P_HIDDEN_BLOCKS, i)
      tubes = innerBool(tubes, P_TUBES, i)
      levelDifficulty = innerInt(levelDifficulty, P_DIFFICULTY, i)
      levelLength = innerInt(levelLength, P_LENGTH, i)
      levelType = innerInt(levelType, P_TYPE, i)
      startMode = innerInt(startMode, P_START_MODE, i)
      timeLimit = innerInt(timeLimit, P_TIME_LIMIT, i)
    }
  }
}

```

```

    buildUpdateFunction(
      Seq((blocks, blocksFn),
        (cannons, cannonsFn),
        (coins, coinsFn),
        (deadEnds, deadEndsFn),
        (enemies, enemiesFn),
        (flatLevel, flatLevelFn),
        (frozenCreatues, frozenFn),
        (pits, pitsFn),
        (hiddenBlocks, hiddenBlocksFn),
        (tubes, tubesFn)),
      Seq((levelDifficulty, difficultyFn),
        (levelLength, lengthFn),
        (levelType, typeFn),
        (startMode, startModeFn),
        (timeLimit, timeLimitFn))
    )
  }

  private def getBooleanOption(pd: ParameterDatabaseInf, base: Parameter, key: String): Option[Boolean] = {
    if (pd.exists(base.push(key), null))
      Some(pd.getBoolean(base.push(key), null, false))
    else
      None
  }

  private def getIntOption(pd: ParameterDatabaseInf, base: Parameter, key: String): Option[Int] = {
    if (pd.exists(base.push(key), null)) {
      Some(pd.getIntWithDefault(base.push(key), null, 0))
    } else {
      None
    }
  }

  def buildUpdateFunction(
    boolSeq: Seq[(Option[Map[Int, Boolean]], (Boolean, MWLevelOptions) => MWLevelOptions)],
    intSeq: Seq[(Option[Map[Int, Int]], (Int, MWLevelOptions) => MWLevelOptions)]
  ): (Int, MWLevelOptions) => MWLevelOptions = {
    def innerBoolLoop(i: Int, opt: MWLevelOptions, boolSeq:
      Seq[(Option[Map[Int, Boolean]], (Boolean, MWLevelOptions) => MWLevelOptions)]): MWLevelOptions = boolSeq
      match {
        case Nil => opt
        case (mapOpt, fn) +: tl => mapOpt match {
          case Some(map) => map.get(i) match {
            case Some(b: Boolean) => innerBoolLoop(i, fn(b, opt), tl)
            case None => innerBoolLoop(i, opt, tl)
          }
          case None => innerBoolLoop(i, opt, tl)
        }
      }

    def innerIntLoop(i: Int, opt: MWLevelOptions, intSeq:
      Seq[(Option[Map[Int, Int]], (Int, MWLevelOptions) => MWLevelOptions)]): MWLevelOptions = intSeq match {
      case Nil => opt
      case (mapOpt, fn) +: tl => mapOpt match {
        case Some(map) => map.get(i) match {
          case Some(x: Int) => innerIntLoop(i, fn(x, opt), tl)
          case None => innerIntLoop(i, opt, tl)
        }
        case None => innerIntLoop(i, opt, tl)
      }
    }

    (i: Int, current: MWLevelOptions) => {
      val afterBool = innerBoolLoop(i, current, boolSeq)
      innerIntLoop(i, afterBool, intSeq)
    }
  }
}

```

---

```

package com.montywest.marioai.learning.ec.params

import org.scalatest.BeforeAndAfterAll
import org.scalamock.scalatest.MockFactory
import org.scalatest.Matchers
import org.scalatest.FlatSpec
import ec.util.ParameterDatabaseInf
import ec.util.Parameter
import org.scalamock.matchers.MockParameter

```

```

import com.montywest.marioai.task.MWEvaluationMultipliers
import com.montywest.marioai.task.MWLevelOptions

class EvaluationParamsUtilTest extends FlatSpec with Matchers with MockFactory with BeforeAndAfterAll {

  private var blankParam = new Parameter("")
  var pdStub = stub[ParameterDatabaseInf]

  override def beforeAll {
    pdStub = stub[ParameterDatabaseInf]
  }

  "getLevelSeeds" should "return defaults for blank database" in {
    val base = blankParam.push(EvaluationParamsUtil.P_LEVEL_BASE)
    (pdStub.exists _) when(*, *) returns(false)
    (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil.P_SEED_START), *, 5) returns(5)
    (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil.P_SEED_ADD), *, 7) returns(7)
    (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil.P_SEED_MULT), *, 9) returns(9)
    assertResult((5, 7, 9)) {
      EvaluationParamsUtil.getLevelSeeds(pdStub, base, 5, 7, 9)
    }
  }

  "getLevelSeeds" should "return start, add and mult params from db" in {
    val base = blankParam.push(EvaluationParamsUtil.P_LEVEL_BASE)
    (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil.P_SEED_START), *, *) returns(2)
    (pdStub.exists _) when(base.push(EvaluationParamsUtil.P_SEED_START), *) returns(true)
    (pdStub.getInt _) when(base.push(EvaluationParamsUtil.P_SEED_START), *, *) returns(2)
    (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil.P_SEED_ADD), *, *) returns(3)
    (pdStub.exists _) when(base.push(EvaluationParamsUtil.P_SEED_ADD), *) returns(true)
    (pdStub.getInt _) when(base.push(EvaluationParamsUtil.P_SEED_ADD), *, *) returns(3)
    (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil.P_SEED_MULT), *, *) returns(4)
    (pdStub.exists _) when(base.push(EvaluationParamsUtil.P_SEED_MULT), *) returns(true)
    (pdStub.getInt _) when(base.push(EvaluationParamsUtil.P_SEED_MULT), *, *) returns(4)

    assertResult((2, 3, 4)) {
      EvaluationParamsUtil.getLevelSeeds(pdStub, base, 5, 7, 9)
    }
  }

  "getNumberOfLevels" should "return None for blank database" in {
    val base = blankParam.push(EvaluationParamsUtil.P_LEVEL_BASE)
    (pdStub.exists _) when(*, *) returns(false)

    assertResult(None) {
      EvaluationParamsUtil.getNumberOfLevels(pdStub, base)
    }
  }

  "getNumberOfLevels" should "return Some(val) for with val from db" in {
    val base = blankParam.push(EvaluationParamsUtil.P_LEVEL_BASE)
    (pdStub.exists _) when(base.push(EvaluationParamsUtil.P_NUM_OF_LEVELS), *) returns(true)
    (pdStub.getInt _) when(base.push(EvaluationParamsUtil.P_NUM_OF_LEVELS), *, *) returns(10)
    (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil.P_NUM_OF_LEVELS), *, *) returns(10)

    assertResult(Some(10)) {
      EvaluationParamsUtil.getNumberOfLevels(pdStub, base)
    }
  }

  "getEvaluationMutlipliers" should "return zero multiplier for blank db" in {
    val base = blankParam.push(EvaluationParamsUtil.P_EVAL_BASE)
    (pdStub.exists _) when(*, *) returns(false)

    assertResult(MWEvaluationMultipliers.zeroEvaluationMultipliers) {
      EvaluationParamsUtil.getEvaluationMutlipliers(pdStub, base)
    }
  }

  "getEvaluationMutlipliers" should "return eval mults from databade" in {
    val base = blankParam.push(EvaluationParamsUtil.P_EVAL_BASE)
    (pdStub.exists _) when(base, *) returns(true)

    (pdStub.exists _) when(base.push(EvaluationParamsUtil.P_COINS), *) returns(true)
    (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil.P_COINS), *, *) returns(10)

    (pdStub.exists _) when(base.push(EvaluationParamsUtil.P_KILLED_BY_SHELL), *) returns(true)
    (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil.P_KILLED_BY_SHELL), *, *) returns(200)

    (pdStub.exists _) when(base.push(EvaluationParamsUtil.P_DISTANCE), *) returns(true)
    (pdStub.getIntWithDefault _) when(base.push(EvaluationParamsUtil.P_DISTANCE), *, *) returns(1)

    val evalMults = EvaluationParamsUtil.getEvaluationMutlipliers(pdStub, base);
  }
}

```

```

    assert(evalMults.coins == 10)
    assert(evalMults.killedByShell == 200)
    assert(evalMults.distance == 1)
    assert(evalMults.win == 0)
    assert(evalMults.kills == 0)
}

"getBaseLevelOptions" should "return defaultOptions for blank db" in {
    val base = blankParam.push(EvaluationParamsUtil.P_LEVEL_BASE)
    (pdStub.exists _) when(*, *) returns(false)

    assertResult(MWLevelOptions.defaultOptions) {
        EvaluationParamsUtil.getBaseLevelOptions(pdStub, base)
    }
}

"getBaseLevelOptions" should "return params from db" in {
    val base = blankParam.push(EvaluationParamsUtil.P_LEVEL_BASE)
    val optBase = base.push(EvaluationParamsUtil.P_BASE_OPTIONS)
    (pdStub.exists _) when(base, *) returns(true)

    (pdStub.exists _) when(optBase.push(EvaluationParamsUtil.P_BLOCKS), *) returns(true)
    (pdStub.getBoolean _) when(optBase.push(EvaluationParamsUtil.P_BLOCKS), *, *) returns(false)

    (pdStub.exists _) when(optBase.push(EvaluationParamsUtil.P_LENGTH), *) returns(true)
    (pdStub.getIntWithDefault _) when(optBase.push(EvaluationParamsUtil.P_LENGTH), *, *) returns(100)

    (pdStub.exists _) when(optBase.push(EvaluationParamsUtil.P_PITS), *) returns(true)
    (pdStub.getBoolean _) when(optBase.push(EvaluationParamsUtil.P_PITS), *, *) returns(false)

    val baseOpt = EvaluationParamsUtil.getBaseLevelOptions(pdStub, base);
    val defaultOpt = MWLevelOptions.defaultOptions

    assert(baseOpt.blocks == false)
    assert(baseOpt.levelLength == 100)
    assert(baseOpt.pits == false)
    assert(baseOpt.tubes == defaultOpt.tubes)
    assert(baseOpt.startingMarioMode == defaultOpt.startingMarioMode)
    assert(baseOpt.timeLimit == defaultOpt.timeLimit)
}

"getUpdateLevelOptionsFunction" should "return trivial lambda for blank database" in {
    val base = blankParam.push(EvaluationParamsUtil.P_LEVEL_BASE)
    (pdStub.exists _) when(*, *) returns(false)

    val updateOpt = EvaluationParamsUtil.getUpdateLevelOptionsFunction(pdStub, base, 3)
    val defaultOpt = MWLevelOptions.defaultOptions

    assert(updateOpt(0, defaultOpt) == defaultOpt)
    assert(updateOpt(1, defaultOpt) == defaultOpt)
    assert(updateOpt(2, defaultOpt) == defaultOpt)
}

"getUpdateLevelOptionsFunction" should "return param lambda from database" in {
    val base = blankParam.push(EvaluationParamsUtil.P_LEVEL_BASE)
    (pdStub.exists _) when(base, *) returns(true)

    (pdStub.exists _) when(base.push(""+0).push(EvaluationParamsUtil.P_BLOCKS), *) returns(true)
    (pdStub.getBoolean _) when(base.push(""+0).push(EvaluationParamsUtil.P_BLOCKS), *, *) returns(false)

    (pdStub.exists _) when(base.push(""+1).push(EvaluationParamsUtil.P_LENGTH), *) returns(true)
    (pdStub.getIntWithDefault _) when(base.push(""+1).push(EvaluationParamsUtil.P_LENGTH), *, *) returns(100)

    (pdStub.exists _) when(base.push(""+2).push(EvaluationParamsUtil.P_PITS), *) returns(true)
    (pdStub.getBoolean _) when(base.push(""+2).push(EvaluationParamsUtil.P_PITS), *, *) returns(false)

    val updateOpt = EvaluationParamsUtil.getUpdateLevelOptionsFunction(pdStub, base, 3)
    val defaultOpt = MWLevelOptions.defaultOptions

    assert(updateOpt(0, defaultOpt).blocks == false)
    assert(updateOpt(1, defaultOpt).levelLength == 100)
    assert(updateOpt(2, defaultOpt).pits == false)
    assert(updateOpt(3, defaultOpt) == defaultOpt)
}
}

```

## F.4 ECJ Modifications

```

/*
Copyright 2006 by Sean Luke and George Mason University

```

```

Licensed under the Academic Free License version 3.0
See the file "LICENSE" for more information
*/

package ec.vector;

import ec.util.*;
import java.io.*;
import ec.*;

/*
 * VectorSpecies.java
 *
 * Created: Thu Mar 22 17:44:00 2001
 * By: Liviu Panait
 */

/**
 * VectorSpecies is a species which can create VectorIndividuals. Different
 * VectorSpecies are used for different kinds of VectorIndividuals: a plain
 * VectorSpecies is probably only applicable for BitVectorIndividuals.
 *
 * <p>VectorSpecies supports the following recombination methods:</p>
 * <ul>
 * <li><b>One-point crossover</b></li>
 * <li><b>Two-point crossover</b></li>
 * <li><b>Uniform crossover</b> - inaccurately called "any-point"</li>
 * <li><b>Line recombination</b> - children are random points on a line between
 * the two parents.</li>
 * <li><b>Intermediate recombination</b> - the value of each component of the
 * vector is between the values of that component of the parent vectors.
 * </li>
 * </ul>
 *
 * <p>Note that BitVectorIndividuals (which use VectorSpecies) and GeneVectorIndividuals
 * (which use GeneVectorSpecies, a subclass of VectorSpecies) do not support
 * Line or Intermediate Recombination.
 *
 * <p>Also note that for LongVectorIndividuals, there are certain values that will
 * never be created by line and intermediate recombination, because the
 * recombination is calculated using doubles and then rounded to the nearest
 * long. For large enough values (but still smaller than the maximum long), the
 * difference between one double and the next is greater than one.</p>
 *
 * <p>VectorSpecies has three ways to determine the initial size of the individual:</p>
 * <ul>
 * <li><b>A fixed size</b></li>
 * <li><b>Geometric distribution</b></li>
 * <li><b>Uniform distribution</b></li>
 * </ul>
 *
 * <p>If the algorithm used is the geometric distribution, the VectorSpecies starts at a
 * minimum size and continues flipping a coin with a certain "resize probability",
 * increasing the size each time, until the coin comes up tails (fails). The chunk size
 * must be 1 in this case.
 *
 * <p>If the algorithm used is the uniform distribution, the VectorSpecies picks a random
 * size between a provided minimum and maximum size, inclusive. The chunk size
 * must be 1 in this case.
 *
 * <p>If the size is fixed, then you can also provide a "chunk size" which constrains the
 * locations in which crossover can be performed (only along chunk boundaries). The genome
 * size must be a multiple of the chunk size in this case.
 *
 * <p>VectorSpecies also contains a number of parameters guiding how the individual
 * crosses over and mutates.
 *
 * <p><b>Per-Gene and Per-Segment Specification.</b> VectorSpecies and its subclasses
 * specify a lot of parameters, notably mutation and initialization parameters, in one
 * of three ways. We will use the <b><tt>mutation-probability</tt></b>
 * parameter as an example.
 *
 * <ol>
 * <li>Globally for all genes in the genome.
 * This is done by specifying:
 * <p><i>base</i><tt>mutation-probability</tt>
 * <br><i>base</i><tt>max-gene</tt>
 * <p><i>Note:</i> you <b>must</b> provide these values even if you don't use them,
 * as they're used as defaults by #2 and #3 below.
 *
 * <p>
 * <li>You may provide parameters for genes in segments (regions) along
 * the genome. The idea is to allow you to specify large chunks of genes
 * all having the same parameter features.

```



```

* To do this you must first specify how many segments there are:
* <p><i>base</i>.<tt>num-segments</tt>
* <p>The segments then may be defined by either start or end indices of genes.
* This is controlled by specifying the value of:
* <p><i>base</i>.<tt>segment-type</tt>
* <p>...which can assume the value of start or end, with start being the default.
* The indices are defined using Java array style, i.e. the first gene has the index of 0,
* and the last gene has the index of genome-size - 1.
* <p>Using this method, each segment is specified by<i>j</i>...
* <p><i>base</i>.<tt>segment.</tt><i>j</i><tt>.start</tt>
* <br><i>base</i>.<tt>segment.</tt><i>j</i><tt>.mutation-probability</tt>
* if segment-type value was chosen as start or by:
* <p><i>base</i>.<tt>segment.</tt><i>j</i><tt>.end</tt>
* <br><i>base</i>.<tt>segment.</tt><i>j</i><tt>.mutation-probability</tt>
* if segment-type value is equal to end.
* <p>
* <li> You may parameters for each separate gene.
* This is done by specifying (for each gene location <i>i</i> you wish to specify)
* <p><i>base</i>.<tt>mutation-probability</tt>.<i>i</i>
* </ol>
*
* <p>Any settings for #3 override #2, and both override #1.
*
* <p>The only parameter which can be specified this way in VectorSpecies is at present
* <tt>mutation-probability</tt>. However a number of parameters are specified this way
* in subclasses.

<p><b>Parameters</b><br>
<table>
<tr><td valign=top><i>base</i>.<tt>genome-size</tt><br>
<font size=-1>int &gt;= 1 or one of: geometric, uniform</font></td>
<td valign=top>(size of the genome, or if 'geometric' or 'uniform', the algorithm used to size the initial
genome)</td></tr>

<tr><td valign=top><i>base</i>.<tt>chunk-size</tt><br>
<font size=-1>int &lt;= genome-size (default=1)</font></td>
<td valign=top>(the chunk size for crossover (crossover will only occur on chunk boundaries))</td></tr>

<tr><td valign=top><i>base</i>.<tt>geometric-prob</tt><br>
<font size=-1>0.0 &lt;= double &lt;= 1.0</font></td>
<td valign=top>(the coin-flip probability for increasing the initial size using the geometric distribution)</td>
</tr>

<tr><td valign=top><i>base</i>.<tt>min-initial-size</tt><br>
<font size=-1>int &gt;= 0</font></td>
<td valign=top>(the minimum initial size of the genome)</td></tr>

<tr><td valign=top><i>base</i>.<tt>max-initial-size</tt><br>
<font size=-1>int &gt;= min-initial-size</font></td>
<td valign=top>(the maximum initial size of the genome)</td></tr>

<tr><td valign=top><i>base</i>.<tt>crossover-type</tt><br>
<font size=-1>string, one of: one, two, any</font></td>
<td valign=top>(default crossover type (one-point, one-point-nonempty, two-point, two-point-nonempty, any-point
(uniform), line, or intermediate)</td></tr>

<tr><td valign=top><i>base</i>.<tt>crossover-prob</tt><br>
<font size=-1>0.0 &gt;= double &lt;= 1.0 </font></td>
<td valign=top>(probability that a gene will get crossed over during any-point (uniform) or simulated binary
crossover)</td></tr>

<tr><td valign=top><i>base</i>.<tt>line-extension</tt><br>
<font size=-1>double &gt;= 0.0 </font></td>
<td valign=top>(for line and intermediate recombination, how far along the line or outside of the hypercube
children can be. If this value is zero, all children must be within the hypercube.)

<tr><td>&nbsp;</td>

<tr><td valign=top style="white-space: nowrap"><i>base</i>.<tt>mutation-prob</tt>&nbsp;&nbsp;&nbsp;<i>or</i><br>
<tr><td valign=top style="white-space: nowrap"><i>base</i>.<tt>segment</tt>.<i>segment-number</i>.<tt>mutation-
prob</tt>&nbsp;&nbsp;&nbsp;<i>or</i><br>
<tr><td valign=top style="white-space: nowrap"><i>base</i>.<tt>mutation-prob</tt>.<i>gene-number</i><br>
<font size=-1>0.0 &lt;= double &lt;= 1.0 </font></td>
<td valign=top>(probability that a gene will get mutated over default mutation)</td></tr>

</table>

<p><b>Default Base</b><br>
vector.species

* @author Sean Luke and Liviu Panait
* @version 1.0
*/

```

```

public class VectorSpecies extends Species
{
    public static final String P_VECTORSPECIES = "species";

    public final static String P_CROSSOVERTYPE = "crossover-type";
    public final static String P_CHUNKSIZE = "chunk-size";
    public final static String V_ONE_POINT = "one";
    public final static String V_ONE_POINT_NO_NOP = "one-nonempty";
    public final static String V_TWO_POINT = "two";
    public final static String V_TWO_POINT_NO_NOP = "two-nonempty";
    public final static String V_ANY_POINT = "any";
    public final static String V_LINE_RECOMB = "line";
    public final static String V_INTERMED_RECOMB = "intermediate";
    public final static String V_SIMULATED_BINARY = "sbx";
    public final static String P_CROSSOVER_DISTRIBUTION_INDEX = "crossover-distribution-index";
    public final static String P_MUTATIONPROB = "mutation-prob";
    public final static String P_CROSSOVERPROB = "crossover-prob";
    public final static String P_GENOMESIZE = "genome-size";
    public final static String P_LINEDISTANCE = "line-extension";
    public final static String V_GEOMETRIC = "geometric";
    public final static String P_GEOMETRIC_PROBABILITY = "geometric-prob";
    public final static String V_UNIFORM = "uniform";
    public final static String P_UNIFORM_MIN = "min-initial-size";
    public final static String P_UNIFORM_MAX = "max-initial-size";
    public final static String P_NUM_SEGMENTS = "num-segments";
    public final static String P_SEGMENT_TYPE = "segment-type";
    public final static String P_SEGMENT_START = "start";
    public final static String P_SEGMENT_END = "end";
    public final static String P_SEGMENT = "segment";
    public final static String P_MODULO_NUM = "modulo-num";
    public final static String P_MODULO = "modulo";
    public final static String P_DUPLICATE_RETRIES = "duplicate-retries";

    public final static int C_ONE_POINT = 0;
    public final static int C_ONE_POINT_NO_NOP = 2;
    public final static int C_TWO_POINT = 4;
    public final static int C_TWO_POINT_NO_NOP = 8;
    public final static int C_ANY_POINT = 128;
    public final static int C_LINE_RECOMB = 256;
    public final static int C_INTERMED_RECOMB = 512;
    public final static int C_SIMULATED_BINARY = 1024;

    public final static int C_NONE = 0;
    public final static int C_GEOMETRIC = 1;
    public final static int C_UNIFORM = 2;

    /** How often do we retry until we get a non-duplicate gene? */
    protected int[] duplicateRetries;

    /** Probability that a gene will mutate, per gene.
     *   This array is one longer than the standard genome length.
     *   The top element in the array represents the parameters for genes in
     *   genomes which have extended beyond the genome length. */
    protected double[] mutationProbability;

    /** Probability that a gene will cross over -- ONLY used in V_ANY_POINT crossover */
    public double crossoverProbability;
    /** What kind of crossover do we have? */
    public int crossoverType;
    /** How big of a genome should we create on initialization? */
    public int genomeSize;
    /** What should the SBX distribution index be? */
    public int crossoverDistributionIndex;
    /** How should we reset the genome? */
    public int genomeResizeAlgorithm;
    /** What's the smallest legal genome? */
    public int minInitialSize;
    /** What's the largest legal genome? */
    public int maxInitialSize;
    /** With what probability would our genome be at least 1 larger than it is now during initialization? */
    public double genomeIncreaseProbability;
    /** How big of chunks should we define for crossover? */
    public int chunksize;
    /** How far along the long a child can be located for line or intermediate recombination */
    public double lineDistance;
    /** Was the initial size determined dynamically? */
    public boolean dynamicInitialSize = false;

    public double mutationProbability(int gene)
    {
        double[] m = mutationProbability;
        if (m.length <= gene)

```

```

        gene = m.length - 1;
    return m[gene];
}

public int duplicateRetries(int gene)
{
    int[] m = duplicateRetries;
    if (m.length <= gene)
        gene = m.length - 1;
    return m[gene];
}

public Parameter defaultBase()
{
    return VectorDefaults.base().push(P_VECTORSPECIES);
}

protected void setupGenome(final EvolutionState state, final Parameter base)
{
    Parameter def = defaultBase();

    String genomeSizeForm = state.parameters.getString(base.push(P_GENOMESIZE), def.push(P_GENOMESIZE));
    if (genomeSizeForm == null) // clearly an error
    {
        state.output.fatal("No genome size specified.", base.push(P_GENOMESIZE), def.push(P_GENOMESIZE));
    }
    else if (genomeSizeForm.equals(V_GEOMETRIC))
    {
        dynamicInitialSize = true;
        genomeSize = 1;
        genomeResizeAlgorithm = C_GEOMETRIC;
        chunksize = state.parameters.getIntWithDefault(base.push(P_CHUNKSIZE), def.push(P_CHUNKSIZE), 1);
        if (chunksize != 1)
            state.output.fatal("To use Geometric size initialization, VectorSpecies must have a chunksize of 1",
                base.push(P_CHUNKSIZE), def.push(P_CHUNKSIZE));
        minInitialSize = state.parameters.getInt(base.push(P_UNIFORM_MIN), def.push(P_UNIFORM_MIN), 0);
        if (minInitialSize < 0)
        {
            state.output.warning("Gemoetric size initialization used, but no minimum initial size provided.
                Assuming minimum is 0.");
            minInitialSize = 0;
        }
        genomeIncreaseProbability = state.parameters.getDoubleWithMax(base.push(P_GEOMETRIC_PROBABILITY), def.
            push(P_GEOMETRIC_PROBABILITY), 0.0, 1.0);
        if (genomeIncreaseProbability < 0.0 || genomeIncreaseProbability >= 1.0) // note >=
            state.output.fatal("To use Gemoetric size initialization, the genome increase probability must be
                >= 0.0 and < 1.0",
                base.push(P_GEOMETRIC_PROBABILITY), def.push(P_GEOMETRIC_PROBABILITY));
    }
    else if (genomeSizeForm.equals(V_UNIFORM))
    {
        dynamicInitialSize = true;
        genomeSize = 1;
        genomeResizeAlgorithm = C_UNIFORM;
        chunksize = state.parameters.getIntWithDefault(base.push(P_CHUNKSIZE), def.push(P_CHUNKSIZE), 1);
        if (chunksize != 1)
            state.output.fatal("To use Uniform size initialization, VectorSpecies must have a chunksize of 1",
                base.push(P_CHUNKSIZE), def.push(P_CHUNKSIZE));
        minInitialSize = state.parameters.getInt(base.push(P_UNIFORM_MIN), def.push(P_UNIFORM_MIN), 0);
        if (minInitialSize < 0)
            state.output.fatal("To use Uniform size initialization, you must set a minimum initial size >= 0",
                base.push(P_UNIFORM_MIN), def.push(P_UNIFORM_MIN));
        maxInitialSize = state.parameters.getInt(base.push(P_UNIFORM_MAX), def.push(P_UNIFORM_MAX), 0);
        if (maxInitialSize < 0)
            state.output.fatal("To use Uniform size initialization, you must set a maximum initial size >= 0",
                base.push(P_UNIFORM_MAX), def.push(P_UNIFORM_MAX));
        if (maxInitialSize < minInitialSize)
            state.output.fatal("To use Uniform size initialization, you must set a maximum initial size >= the
                minimum initial size",
                base.push(P_UNIFORM_MAX), def.push(P_UNIFORM_MAX));
    }
    else // it's a number
    {
        genomeSize = state.parameters.getInt(base.push(P_GENOMESIZE), def.push(P_GENOMESIZE), 1);
        if (genomeSize==0)
            state.output.fatal("VectorSpecies must have a genome size > 0",
                base.push(P_GENOMESIZE), def.push(P_GENOMESIZE));

        genomeResizeAlgorithm = C_NONE;

        chunksize = state.parameters.getIntWithDefault(base.push(P_CHUNKSIZE), def.push(P_CHUNKSIZE), 1);
        if (chunksize <= 0 || chunksize > genomeSize)

```

```

        state.output.fatal("VectorSpecies must have a chunksize which is > 0 and < genomeSize",
            base.push(P_CHUNKSIZE), def.push(P_CHUNKSIZE));
        if (genomeSize % chunksize != 0)
            state.output.fatal("VectorSpecies must have a genomeSize which is a multiple of chunksize",
                base.push(P_CHUNKSIZE), def.push(P_CHUNKSIZE));
    }
}

public void setup(final EvolutionState state, final Parameter base)
{
    Parameter def = defaultBase();

    // We will construct, but NOT set up, a sacrificial individual here.
    // Actual setup is done at the end of this method (in super.setup(...))
    // The purpose of this sacrificial individual is to enable methods such
    // as inNumericalTypeRange() to run properly, since they require knowledge
    // of which KIND of individual it is.

    i_prototype = (Individual)(state.parameters.getInstanceForParameter(
        base.push(P_INDIVIDUAL), def.push(P_INDIVIDUAL),
        Individual.class));

    // this will get thrown away and replaced with a new one during super.setup(...).

    // this might get called twice, I don't think it's a big deal
    setupGenome(state, base);

    // MUTATION

    double _mutationProbability = state.parameters.getDoubleWithMax(base.push(P_MUTATIONPROB), def.push(
        P_MUTATIONPROB), 0.0, 1.0);
    if (_mutationProbability == -1.0)
        state.output.fatal("Global mutation probability must be between 0.0 and 1.0 inclusive",
            base.push(P_MUTATIONPROB), def.push(P_MUTATIONPROB));
    mutationProbability = fill(new double[genomeSize + 1], _mutationProbability);

    int _duplicateRetries = state.parameters.getIntWithDefault(base.push(P_DUPLICATE_RETRIES), def.push(
        P_DUPLICATE_RETRIES), 0);
    if (_duplicateRetries < 0)
    {
        state.output.fatal("Duplicate Retries, if defined, must be a value >= 0", base.push(
            P_DUPLICATE_RETRIES), def.push(P_DUPLICATE_RETRIES));
    }
    duplicateRetries = fill(new int[genomeSize + 1], _duplicateRetries);

    // CROSSOVER

    String ctype = state.parameters.getStringWithDefault(base.push(P_CROSSOVERTYPE), def.push(P_CROSSOVERTYPE),
        null);
    crossoverType = C_ONE_POINT;
    if (ctype==null)
        state.output.warning("No crossover type given for VectorSpecies, assuming one-point crossover",
            base.push(P_CROSSOVERTYPE), def.push(P_CROSSOVERTYPE));
    else if (ctype.equalsIgnoreCase(V_ONE_POINT))
        crossoverType=C_ONE_POINT; // redundant
    else if (ctype.equalsIgnoreCase(V_ONE_POINT_NO_NOP))
        crossoverType=C_ONE_POINT_NO_NOP;
    else if (ctype.equalsIgnoreCase(V_TWO_POINT))
        crossoverType=C_TWO_POINT;
    else if (ctype.equalsIgnoreCase(V_TWO_POINT_NO_NOP))
        crossoverType=C_TWO_POINT_NO_NOP;
    else if (ctype.equalsIgnoreCase(V_ANY_POINT))
        crossoverType=C_ANY_POINT;
    else if (ctype.equalsIgnoreCase(V_LINE_RECOMB))
        crossoverType=C_LINE_RECOMB;
    else if (ctype.equalsIgnoreCase(V_INTERMED_RECOMB))
        crossoverType=C_INTERMED_RECOMB;
    else if (ctype.equalsIgnoreCase(V_SIMULATED_BINARY))
        crossoverType=C_SIMULATED_BINARY;
    else state.output.fatal("VectorSpecies given a bad crossover type: " + ctype,
        base.push(P_CROSSOVERTYPE), def.push(P_CROSSOVERTYPE));

    if (crossoverType==C_LINE_RECOMB || crossoverType==C_INTERMED_RECOMB)
    {
        if (!(this instanceof IntegerVectorSpecies) && !(this instanceof FloatVectorSpecies))
            state.output.fatal("Line and intermediate recombinations are only supported by
                IntegerVectorSpecies and FloatVectorSpecies", base.push(P_CROSSOVERTYPE), def.push(
                    P_CROSSOVERTYPE));
    }
}

```

```

        lineDistance = state.parameters.getDouble(
            base.push(P_LINEDISTANCE), def.push(P_LINEDISTANCE), 0.0);
        if (lineDistance==1.0)
            state.output.fatal("If it's going to use line or intermediate recombination, VectorSpecies needs a
                line extension >= 0.0 (0.25 is common)", base.push(P_LINEDISTANCE), def.push(P_LINEDISTANCE
                ));
        }
    else lineDistance = 0.0;

    if (crossoverType==C_ANY_POINT)
    {
        crossoverProbability = state.parameters.getDoubleWithMax(
            base.push(P_CROSSOVERPROB), def.push(P_CROSSOVERPROB), 0.0, 0.5);
        if (crossoverProbability==1.0)
            state.output.fatal("If it's going to use any-point crossover, VectorSpecies must have a crossover
                probability between 0.0 and 0.5 inclusive",
                base.push(P_CROSSOVERPROB), def.push(P_CROSSOVERPROB));
        }
    else if (crossoverType==C_SIMULATED_BINARY)
    {
        if (!(this instanceof FloatVectorSpecies))
            state.output.fatal("Simulated binary crossover (SBX) is only supported by FloatVectorSpecies",
                base.push(P_CROSSOVERTYPE), def.push(P_CROSSOVERTYPE));
        crossoverDistributionIndex = state.parameters.getInt(base.push(P_CROSSOVER_DISTRIBUTION_INDEX), def.
            push(P_CROSSOVER_DISTRIBUTION_INDEX), 0);
        if (crossoverDistributionIndex < 0)
            state.output.fatal("If FloatVectorSpecies is going to use simulated binary crossover (SBX), the
                distribution index must be defined and >= 0.",
                base.push(P_CROSSOVER_DISTRIBUTION_INDEX), def.push(P_CROSSOVER_DISTRIBUTION_INDEX));
        }
    else crossoverProbability = 0.0;

    state.output.exitIfErrors();

    if (crossoverType != C_ANY_POINT && state.parameters.exists(base.push(P_CROSSOVERPROB), def.push(
        P_CROSSOVERPROB)))
        state.output.warning("The 'crossover-prob' parameter may only be used with any-point crossover. It
            states the probability that a particular gene will be crossed over. If you were looking for the
            probability of crossover happening at *all*, look at the 'likelihood' parameter.",
            base.push(P_CROSSOVERPROB), def.push(P_CROSSOVERPROB));

/**
 * Edit by Monty West
 */
// MODULO VALUES
if (state.parameters.exists(base.push(P_MODULO_NUM), def.push(P_MODULO_NUM)))
{
    if (dynamicInitialSize)
        state.output.warnOnce("Using dynamic initial sizing, but modulo min/max gene declarations. This is
            probably wrong. You probably want to use global min/max declarations.",
            base.push(P_MODULO_NUM), def.push(P_MODULO_NUM));

    int moduloNum = state.parameters.getIntWithDefault(base.push(P_MODULO_NUM), def.push(P_MODULO_NUM)
        , 0);

    if (moduloNum == 0)
        state.output.fatal(
            "Modulo by zero is undefined. " + P_MODULO_NUM + " must be non-zero.",
            base.push(P_MODULO_NUM),
            def.push(P_MODULO_NUM));

    if ((genomeSize % moduloNum) != 0)
        state.output.fatal(
            P_MODULO_NUM + " must divide genome size.",
            base.push(P_MODULO_NUM),
            def.push(P_MODULO_NUM));

    initializeGenomeByModuloIndices(state, base, def, moduloNum);
}

state.output.exitIfErrors();

// SEGMENTS

// Set number of segments to 0 by default
int numSegments = 0;
// Now check to see if segments of genes (genes having the same min and
// max values) exist
if (state.parameters.exists(base.push(P_NUM_SEGMENTS), def.push(P_NUM_SEGMENTS)))
{

```

```

        if (dynamicInitialSize)
            state.output.warnOnce("Using dynamic initial sizing, but per-segment min/max gene declarations.
                This is probably wrong. You probably want to use global min/max declarations.",
                base.push(P_NUM_SEGMENTS), def.push(P_NUM_SEGMENTS));

        numSegments = state.parameters.getIntWithDefault(base.push(P_NUM_SEGMENTS),
            def.push(P_NUM_SEGMENTS), 0);

        if (numSegments == 0)
            state.output.warning(
                "The number of genome segments has been defined to be equal to 0.\n"
                + "Hence, no genome segments will be defined.",
                base.push(P_NUM_SEGMENTS),
                def.push(P_NUM_SEGMENTS));
        else if (numSegments < 0)
            state.output.fatal(
                "Invalid number of genome segments: " + numSegments
                + "\nIt must be a nonnegative value.",
                base.push(P_NUM_SEGMENTS),
                def.push(P_NUM_SEGMENTS));

        //read the type of segment definition using the default start value
        String segmentType = state.parameters.getStringWithDefault(base.push(P_SEGMENT_TYPE),
            def.push(P_SEGMENT_TYPE), P_SEGMENT_START);

        if (segmentType.equalsIgnoreCase(P_SEGMENT_START))
            initializeGenomeSegmentsByStartIndices(state, base, def, numSegments);
        else if (segmentType.equalsIgnoreCase(P_SEGMENT_END))
            initializeGenomeSegmentsByEndIndices(state, base, def, numSegments);
        else
            state.output.fatal(
                "Invalid specification of genome segment type: " + segmentType
                + "\nThe " + P_SEGMENT_TYPE + " parameter must have the value of " + P_SEGMENT_START + " or "
                + P_SEGMENT_END,
                base.push(P_SEGMENT_TYPE),
                def.push(P_SEGMENT_TYPE));
    }
    state.output.exitIfErrors();

    // PER-GENE VALUES

    for (int x = 0; x < genomeSize; x++)
    {
        loadParametersForGene(state, x, base, def, "" + x);
    }
    state.output.exitIfErrors();

    prePrototypeSetup(state, base, def);
    state.output.exitIfErrors();

    // NOW call super.setup(...), which will in turn set up the prototypical individual
    super.setup(state, base);
}

protected void prePrototypeSetup(EvolutionState state, Parameter base, Parameter def)
{
    //None by default, for subclasses
}

/** Called when VectorSpecies is setting up per-gene and per-segment parameters. The index
    is the current gene whose parameter is getting set up. The Parameters in question are the
    bases for the gene. The postfix should be appended to the end of any parameter looked up
    (it often contains a number indicating the gene in question), such as
    state.parameters.exists(base.push(P_PARAM).push(postfix), def.push(P_PARAM).push(postfix)

    <p>If you override this method, be sure to call super(...) at some point, ideally first.
*/
protected void loadParametersForGene(EvolutionState state, int index, Parameter base, Parameter def, String
    postfix)
{
    // our only per-gene parameter is mutation probability.

    if (state.parameters.exists(base.push(P_MUTATIONPROB).push(postfix), def.push(P_MUTATIONPROB).push(
        postfix)))
    {
        mutationProbability[index] = state.parameters.getDoubleWithMax(base.push(P_MUTATIONPROB).push(postfix)
            , def.push(P_MUTATIONPROB).push(postfix), 0.0, 1.0);
        if (mutationProbability[index] == -1.0)
            state.output.fatal("Per-gene or per-segment mutation probability must be between 0.0 and 1.0
                inclusive",
                base.push(P_MUTATIONPROB).push(postfix), def.push(P_MUTATIONPROB).push(postfix));
    }
}

```

```

        if (state.parameters.exists(base.push(P_DUPLICATE_RETRIES).push(postfix), def.push(P_DUPLICATE_RETRIES).
            push(postfix)))
        {
            duplicateRetries[index] = state.parameters.getInt(base.push(P_DUPLICATE_RETRIES).push(postfix), def.
                push(P_DUPLICATE_RETRIES).push(postfix));
            if (duplicateRetries[index] < 0)
                state.output.fatal("Duplicate Retries for gene " + index + ", if defined must be a value >= 0",
                    base.push(P_DUPLICATE_RETRIES).push(postfix), def.push(P_DUPLICATE_RETRIES).push(postfix));
        }
    }

    protected void initializeGenomeSegmentsByStartIndices(final EvolutionState state,
        final Parameter base,
        final Parameter def,
        int numSegments)
    {
        this.initializeGenomeSegmentsByStartIndices(state, base, def, numSegments, genomeSize, 0, 0);
    }

    /** Looks up genome segments using start indices. Segments run up to the next declared start index. */
    protected void initializeGenomeSegmentsByStartIndices(final EvolutionState state,
        final Parameter base,
        final Parameter def,
        int numSegments,
        int previousSegmentEnd,
        int currentSegmentEnd,
        int expectedStart)
    {
        //loop in reverse order

        for (int i = numSegments - 1; i >= 0; i--)
        {
            //check if the segment data exist
            if (state.parameters.exists(base.push(P_SEGMENT).push(""+i).push(P_SEGMENT_START),
                def.push(P_SEGMENT).push(""+i).push(P_SEGMENT_START)))
            {
                //Read the index of the end gene specifying current segment
                currentSegmentEnd = state.parameters.getInt(base.push(P_SEGMENT).push(""+i).push(P_SEGMENT_START),
                    def.push(P_SEGMENT).push(""+i).push(P_SEGMENT_START));
            }
            else
            {
                state.output.fatal("Genome segment " + i + " has not been defined!" +
                    "\nYou must specify start indices for " + numSegments + " segment(s)",
                    base.push(P_SEGMENT).push(""+i).push(P_SEGMENT_START),
                    base.push(P_SEGMENT).push(""+i).push(P_SEGMENT_START));
            }

            //check if the start index is valid
            if (currentSegmentEnd >= previousSegmentEnd || currentSegmentEnd < expectedStart)
                state.output.fatal(
                    "Invalid start index value for segment " + i + ": " + currentSegmentEnd
                    + "\nThe value must be smaller than " + previousSegmentEnd +
                    " and greater than or equal to " + expectedStart);

            //check if the index of the first segment is equal to 0
            if (i == 0 && currentSegmentEnd != expectedStart)
                state.output.fatal(
                    "Invalid start index value for the first segment " + i + ": " + currentSegmentEnd
                    + "\nThe value must be equal to " + expectedStart);

            //and assign min and max values for all genes in this segment
            for (int j = previousSegmentEnd-1; j >= currentSegmentEnd; j--)
            {
                loadParametersForGene(state, j, base.push(P_SEGMENT).push(""+i), def.push(P_SEGMENT).push(""+i), "
                ");
            }

            previousSegmentEnd = currentSegmentEnd;
        }
    }

    /** Looks up genome segments using end indices. Segments run from the previously declared end index. */
    protected void initializeGenomeSegmentsByEndIndices(final EvolutionState state,
        final Parameter base,
        final Parameter def,
        int numSegments)
    {
        this.initializeGenomeSegmentsByEndIndices(state, base, def, numSegments, -1, 0, genomeSize-1);
    }

```

```

    }

protected void initializeGenomeSegmentsByEndIndices(final EvolutionState state,
    final Parameter base,
    final Parameter def,
    int numSegments,
    int previousSegmentEnd,
    int currentSegmentEnd,
    int expectedEnd)
{
    // iterate over segments and set genes values for each segment
    for (int i = 0; i < numSegments; i++)
    {
        //check if the segment data exist
        if (state.parameters.exists(base.push(P_SEGMENT).push(""+i).push(P_SEGMENT_END), def.push(P_SEGMENT).push(""+i).push(P_SEGMENT_END)))
        {
            //Read the index of the end gene specifying current segment
            currentSegmentEnd = state.parameters.getInt(base.push(P_SEGMENT).push(""+i).push(P_SEGMENT_END), def.push(P_SEGMENT).push(""+i).push(P_SEGMENT_END));
        }
        else
        {
            state.output.fatal("Genome segment " + i + " has not been defined!" +
                "\nYou must specify end indices for " + numSegments + " segment(s)",
                base.push(P_SEGMENT).push(""+i).push(P_SEGMENT_END),
                base.push(P_SEGMENT).push(""+i).push(P_SEGMENT_END));
        }

        //check if the end index is valid
        if(currentSegmentEnd <= previousSegmentEnd || currentSegmentEnd > expectedEnd)
            state.output.fatal(
                "Invalid end index value for segment " + i + ": " + currentSegmentEnd
                + "\nThe value must be greater than " + previousSegmentEnd +
                " and smaller/equal than " + expectedEnd);

        //check if the index of the final segment is equal to the genomeSize
        if(i == numSegments - 1 && currentSegmentEnd != (expectedEnd-1))
            state.output.fatal(
                "Invalid end index value for the last segment " + i + ": " + currentSegmentEnd
                + "\nThe value must be equal to the index of the last gene in the genome: " + (expectedEnd-1))
            ;

        //and assign min and max values for all genes in this segment
        for(int j = previousSegmentEnd+1; j <= currentSegmentEnd; j++)
        {
            loadParametersForGene(state, j, base.push(P_SEGMENT).push(""+i), def.push(P_SEGMENT).push(""+i), "
");
        }

        previousSegmentEnd = currentSegmentEnd;
    }
}

protected void initializeGenomeByModuloIndices(final EvolutionState state,
    final Parameter base,
    final Parameter def,
    int moduloNum)
{
    for (int i = 0; i < moduloNum; i++)
    {
        if (state.parameters.exists(base.push(P_MODULO).push(""+i), def.push(P_MODULO).push(""+i)))
        {
            for(int j = i; j < genomeSize; j = j + moduloNum)
            {
                loadParametersForGene(state, j, base.push(P_MODULO).push(""+i), def.push(
                    P_MODULO).push(""+i), "");
            }
        }
    }
}

public Individual newIndividual(final EvolutionState state, int thread)
{
    VectorIndividual newind = (VectorIndividual)(super.newIndividual(state, thread));

    if (genomeResizeAlgorithm == C_NONE)
        newind.reset( state, thread );
    else if (genomeResizeAlgorithm == C_UNIFORM)

```



```

    {
        int size = state.random[thread].nextInt(maxInitialSize - minInitialSize + 1) + minInitialSize;
        newind.reset(state, thread, size);
    }
    else if (genomeResizeAlgorithm == C_GEOMETRIC)
    {
        int size = minInitialSize;
        while(state.random[thread].nextBoolean(genomeIncreaseProbability)) size++;
        newind.reset(state, thread, size);
    }

    return newind;
}

// These convenience methods are used by subclasses to fill arrays and check to see if
// arrays contain certain values.

/** Utility method: fills the array with the given value and returns it. */
protected long[] fill(long[] array, long val)
{
    for(int i =0; i < array.length; i++) array[i] = val;
    return array;
}

/** Utility method: fills the array with the given value and returns it. */
protected int[] fill(int[] array, int val)
{
    for(int i =0; i < array.length; i++) array[i] = val;
    return array;
}

/** Utility method: fills the array with the given value and returns it. */
protected boolean[] fill(boolean[] array, boolean val)
{
    for(int i =0; i < array.length; i++) array[i] = val;
    return array;
}

/** Utility method: fills the array with the given value and returns it. */
protected double[] fill(double[] array, double val)
{
    for(int i =0; i < array.length; i++) array[i] = val;
    return array;
}

/** Utility method: returns the first array slot which contains the given value, else -1. */
protected int contains(boolean[] array, boolean val)
{
    for(int i =0; i < array.length; i++)
        if (array[i] == val) return i;
    return -1;
}

/** Utility method: returns the first array slot which contains the given value, else -1. */
protected int contains(long[] array, long val)
{
    for(int i =0; i < array.length; i++)
        if (array[i] == val) return i;
    return -1;
}

/** Utility method: returns the first array slot which contains the given value, else -1. */
protected int contains(int[] array, int val)
{
    for(int i =0; i < array.length; i++)
        if (array[i] == val) return i;
    return -1;
}

/** Utility method: returns the first array slot which contains the given value, else -1. */
protected int contains(double[] array, double val)
{
    for(int i =0; i < array.length; i++)
        if (array[i] == val) return i;
    return -1;
}
}

```

---

```

/*

```

Copyright 2006 by Sean Luke  
 Licensed under the Academic Free License version 3.0  
 See the file "LICENSE" for more information  
 \*/

```
package ec.vector;
import ec.*;
import ec.util.*;
```

```
/*
 * IntegerVectorSpecies.java
 *
 * Created: Tue Feb 20 13:26:00 2001
 * By: Sean Luke
 */
```

```
/**
 * IntegerVectorSpecies is a subclass of VectorSpecies with special constraints
 * for integral vectors, namely ByteVectorIndividual, ShortVectorIndividual,
 * IntegerVectorIndividual, and LongVectorIndividual.
 *
 * <p>IntegerVectorSpecies can specify a number of parameters globally, per-segment, and per-gene.
 * See <a href="VectorSpecies.html">VectorSpecies</a> for information on how to this works.
 *
 * <p>IntegerVectorSpecies defines a minimum and maximum gene value. These values
 * are used during initialization and, depending on whether <tt>mutation-bounded</tt>
 * is true, also during mutation algorithms to guarantee that the gene value
 * will not exceed these minimum and maximum bounds.
 *
 * <p>
 * IntegerVectorSpecies provides support for two ways of mutating a gene.
 * <ul>
 * <li><b>reset</b> Replacing the gene's value with a value uniformly drawn from the gene's
 * range (the default behavior).</li>
 * <li><b>random-walk</b> Replacing the gene's value by performing a random walk starting at the gene
 * value. The random walk either adds 1 or subtracts 1 (chosen at random), then does a coin-flip
 * to see whether to continue the random walk. When the coin-flip finally comes up false, the gene value
 * is set to the current random walk position.
 * </ul>
 *
 * <p>IntegerVectorSpecies performs gene initialization by resetting the gene.
 *
 * <p><b>Parameters</b><br>
 * <table>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>min-gene</tt> &nbsp;&nbsp;&nbsp;<i>or</i><br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>segment</tt>. <i>segment-number</i>. <tt>min-gene</tt> &nbsp;&nbsp;&nbsp;<i>or</i><br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>min-gene</tt>. <i>gene-number</i><br>
 * <font size=-1>long (default=0)</font></td>
 * <td valign=top>(the minimum gene value)</td></tr>
 *
 * <tr><td>&nbsp;&nbsp;&nbsp;<br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>max-gene</tt> &nbsp;&nbsp;&nbsp;<i>or</i><br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>segment</tt>. <i>segment-number</i>. <tt>max-gene</tt> &nbsp;&nbsp;&nbsp;<i>or</i><br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>max-gene</tt>. <i>gene-number</i><br>
 * <font size=-1>long &gt;= <i>base</i>. <tt>min-gene</tt></font></td>
 * <td valign=top>(the maximum gene value)</td></tr>
 *
 * <tr><td>&nbsp;&nbsp;&nbsp;<br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>mutation-type</tt> &nbsp;&nbsp;&nbsp;<i>or</i><br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>segment</tt>. <i>segment-number</i>. <tt>mutation-
 * type</tt> &nbsp;&nbsp;&nbsp;<i>or</i><br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>mutation-prob</tt>. <i>gene-number</i><br>
 * <font size=-1><tt>reset</tt> or <tt>random-walk</tt> (default=<tt>reset</tt>)</font></td>
 * <td valign=top>(the mutation type)</td>
 * </tr>
 *
 * <tr><td>&nbsp;&nbsp;&nbsp;<br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>random-walk-probability</tt> &nbsp;&nbsp;&nbsp;<i>
 * or</i><br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>segment</tt>. <i>segment-number</i>. <tt>random-
 * walk-probability</tt> &nbsp;&nbsp;&nbsp;<i>or</i><br>
 * <tr><td valign=top style="white-space: nowrap"><i>base</i>. <tt>random-walk-probability</tt>. <i>gene-number</i><br>
 * <font size=-1>0.0 &lt;= double &lt;= 1.0 </font></td>
 * <td valign=top>(the probability that a random walk will continue. Random walks go up or down by 1.0 until the
 * coin flip comes up false.)</td>
 * </tr>
 *
 * <tr><td>&nbsp;&nbsp;&nbsp;<br>
```

```
|  |
| --- |
| <i>base</i>.<tt>mutation-bounded</tt>&nbsp;&nbsp;&nbsp;<i>or</i><br> |
| <i>base</i>.<tt>segment</tt>.<i>segment-number</i>.<tt>mutation-bounded</tt>&nbsp;&nbsp;&nbsp;<i>or</i><br> |
| <i>base</i>.<tt>mutation-bounded</tt>.<i>gene-number</i><br> |

```

\* <font size=-1>boolean (default=true)</font></td>  
\* <td align=top>(whether mutation is restricted to only being within the min/max gene values. Does not apply to SimulatedBinaryCrossover (which is always bounded))</td>  
\* </tr>  
\* </table>  
\* @author Sean Luke, Rafal Kicinger  
\* @version 1.0  
\*/

```

public class IntegerVectorSpecies extends VectorSpecies
{
    public final static String P_MINGENE = "min-gene";
    public final static String P_MAXGENE = "max-gene";

    public final static String P_NUM_SEGMENTS = "num-segments";

    public final static String P_SEGMENT_TYPE = "segment-type";

    public final static String P_SEGMENT_START = "start";

    public final static String P_SEGMENT_END = "end";

    public final static String P_SEGMENT = "segment";

    public final static String P_MUTATIONTYPE = "mutation-type";

    public final static String P_RANDOM_WALK_PROBABILITY = "random-walk-probability";

    public final static String P_MUTATION_BOUNDED = "mutation-bounded";

    public final static String V_RESET_MUTATION = "reset";

    public final static String V_RANDOM_WALK_MUTATION = "random-walk";

    public final static int C_RESET_MUTATION = 0;

    public final static int C_RANDOM_WALK_MUTATION = 1;

    /** Min-gene value, per gene.
     * This array is one longer than the standard genome length.
     * The top element in the array represents the parameters for genes in
     * genomes which have extended beyond the genome length. */
    protected long[] minGene;

    /** Max-gene value, per gene.
     * This array is one longer than the standard genome length.
     * The top element in the array represents the parameters for genes in
     * genomes which have extended beyond the genome length. */
    protected long[] maxGene;

    /** Mutation type, per gene.
     * This array is one longer than the standard genome length.
     * The top element in the array represents the parameters for genes in
     * genomes which have extended beyond the genome length. */
    protected int[] mutationType;

    /** The continuation probability for Integer Random Walk Mutation, per gene.
     * This array is one longer than the standard genome length.
     * The top element in the array represents the parameters for genes in
     * genomes which have extended beyond the genome length. */
    protected double[] randomWalkProbability;

    /** Whether mutation is bounded to the min- and max-gene values, per gene.
     * This array is one longer than the standard genome length.
     * The top element in the array represents the parameters for genes in
     * genomes which have extended beyond the genome length. */
    protected boolean[] mutationIsBounded;

    /** Whether the mutationIsBounded value was defined, per gene.
     * Used internally only.
     * This array is one longer than the standard genome length.
     * The top element in the array represents the parameters for genes in
     * genomes which have extended beyond the genome length. */
    boolean mutationIsBoundedDefined;

    public long maxGene(int gene)
    {

```

```

        long[] m = maxGene;
        if (m.length <= gene)
            gene = m.length - 1;
        return m[gene];
    }

    public long minGene(int gene)
    {
        long[] m = minGene;
        if (m.length <= gene)
            gene = m.length - 1;
        return m[gene];
    }

    public int mutationType(int gene)
    {
        int[] m = mutationType;
        if (m.length <= gene)
            gene = m.length - 1;
        return m[gene];
    }

    public double randomWalkProbability(int gene)
    {
        double[] m = randomWalkProbability;
        if (m.length <= gene)
            gene = m.length - 1;
        return m[gene];
    }

    public boolean mutationIsBounded(int gene)
    {
        boolean[] m = mutationIsBounded;
        if (m.length <= gene)
            gene = m.length - 1;
        return m[gene];
    }

    public boolean inNumericalTypeRange(double geneVal)
    {
        if (i_prototype instanceof ByteVectorIndividual)
            return (geneVal <= Byte.MAX_VALUE && geneVal >= Byte.MIN_VALUE);
        else if (i_prototype instanceof ShortVectorIndividual)
            return (geneVal <= Short.MAX_VALUE && geneVal >= Short.MIN_VALUE);
        else if (i_prototype instanceof IntegerVectorIndividual)
            return (geneVal <= Integer.MAX_VALUE && geneVal >= Integer.MIN_VALUE);
        else if (i_prototype instanceof LongVectorIndividual)
            return true; // geneVal is valid for all longs
        else return false; // dunno what the individual is...
    }

    public boolean inNumericalTypeRange(long geneVal)
    {
        if (i_prototype instanceof ByteVectorIndividual)
            return (geneVal <= Byte.MAX_VALUE && geneVal >= Byte.MIN_VALUE);
        else if (i_prototype instanceof ShortVectorIndividual)
            return (geneVal <= Short.MAX_VALUE && geneVal >= Short.MIN_VALUE);
        else if (i_prototype instanceof IntegerVectorIndividual)
            return (geneVal <= Integer.MAX_VALUE && geneVal >= Integer.MIN_VALUE);
        else if (i_prototype instanceof LongVectorIndividual)
            return true; // geneVal is valid for all longs
        else return false; // dunno what the individual is...
    }

    public void setup(final EvolutionState state, final Parameter base)
    {
        Parameter def = defaultBase();

        setupGenome(state, base);

        // create the arrays
        minGene = new long[genomeSize + 1];
        maxGene = new long[genomeSize + 1];
        mutationType = fill(new int[genomeSize + 1], -1);
        mutationIsBounded = new boolean[genomeSize + 1];
        randomWalkProbability = new double[genomeSize + 1];

        // LOADING GLOBAL MIN/MAX GENES
        long _minGene = state.parameters.getLongWithDefault(base.push(P_MINGENE), def.push(P_MINGENE), 0);
        long _maxGene = state.parameters.getLong(base.push(P_MAXGENE), def.push(P_MAXGENE), _minGene);
        if (_maxGene < _minGene)
            state.output.fatal("IntegerVectorSpecies must have a default min-gene which is <= the default max-gene",
                );
    }

```

```

        base.push(P_MAXGENE), def.push(P_MAXGENE));
fill(minGene, _minGene);
fill(maxGene, _maxGene);

/// MUTATION

String mtype = state.parameters.getStringWithDefault(base.push(P_MUTATIONTYPE), def.push(P_MUTATIONTYPE),
    null);
int _mutationType = C_RESET_MUTATION;
if (mtype == null)
    state.output.warning("No global mutation type given for IntegerVectorSpecies, assuming 'reset'
        mutation",
        base.push(P_MUTATIONTYPE), def.push(P_MUTATIONTYPE));
else if (mtype.equalsIgnoreCase(V_RESET_MUTATION))
    _mutationType = C_RESET_MUTATION; // redundant
else if (mtype.equalsIgnoreCase(V_RANDOM_WALK_MUTATION))
    _mutationType = C_RANDOM_WALK_MUTATION;
else
    state.output.fatal("IntegerVectorSpecies given a bad mutation type: "
        + mtype, base.push(P_MUTATIONTYPE), def.push(P_MUTATIONTYPE));
fill(mutationType, _mutationType);

if (_mutationType == C_RANDOM_WALK_MUTATION)
{
    double _randomWalkProbability = state.parameters.getDoubleWithMax(base.push(P_RANDOM_WALK_PROBABILITY),
        def.push(P_RANDOM_WALK_PROBABILITY), 0.0, 1.0);
    if (_randomWalkProbability <= 0)
        state.output.fatal("If it's going to use random walk mutation as its global mutation type,
            IntegerVectorSpecies must a random walk mutation probability between 0.0 and 1.0.",
            base.push(P_RANDOM_WALK_PROBABILITY), def.push(P_RANDOM_WALK_PROBABILITY));
    fill(randomWalkProbability, _randomWalkProbability);

    if (!state.parameters.exists(base.push(P_MUTATION_BOUNDED), def.push(P_MUTATION_BOUNDED)))
        state.output.warning("IntegerVectorSpecies is using gaussian, polynomial, or integer randomwalk
            mutation as its global mutation type, but " + P_MUTATION_BOUNDED + " is not defined.
            Assuming 'true'");
    boolean _mutationIsBounded = state.parameters.getBoolean(base.push(P_MUTATION_BOUNDED), def.push(
        P_MUTATION_BOUNDED), true);
    fill(mutationIsBounded, _mutationIsBounded);
    mutationIsBoundedDefined = true;
}

super.setup(state, base);

// VERIFY
for(int x=0; x< genomeSize + 1; x++)
{
    if (maxGene[x] < minGene[x])
        state.output.fatal("IntegerVectorSpecies must have a min-gene["+x+"] which is <= the max-gene["+x+
            "];");

    // check to see if these longs are within the data type of the particular individual
    if (!inNumericalTypeRange(minGene[x]))
        state.output.fatal("This IntegerVectorSpecies has a prototype of the kind: "
            + i.prototype.getClass().getName() +
            ", but doesn't have a min-gene["+x+"] value within the range of this prototype's genome's data
            types");
    if (!inNumericalTypeRange(maxGene[x]))
        state.output.fatal("This IntegerVectorSpecies has a prototype of the kind: "
            + i.prototype.getClass().getName() +
            ", but doesn't have a max-gene["+x+"] value within the range of this prototype's genome's data
            types");
}

/*
//Debugging
for(int i = 0; i < minGene.length; i++)
System.out.println("Min: " + minGene[i] + ", Max: " + maxGene[i]);
*/
}

protected void loadParametersForGene(EvolutionState state, int index, Parameter base, Parameter def, String
    postfix)
{
    super.loadParametersForGene(state, index, base, def, postfix);
}

```

```

boolean minValExists = state.parameters.exists(base.push(P_MINGENE).push(postfix), def.push(P_MINGENE).
    push(postfix));
boolean maxValExists = state.parameters.exists(base.push(P_MAXGENE).push(postfix), def.push(P_MAXGENE).
    push(postfix));

if ((maxValExists && !(minValExists)))
    state.output.warning("Max Gene specified but not Min Gene", base.push(P_MINGENE).push(postfix), def.
        push(P_MINGENE).push(postfix));

if ((minValExists && !(maxValExists)))
    state.output.warning("Min Gene specified but not Max Gene", base.push(P_MAXGENE).push(postfix), def.
        push(P_MINGENE).push(postfix));

if (minValExists)
{
    long minVal = state.parameters.getLongWithDefault(base.push(P_MINGENE).push(postfix), def.push(
        P_MINGENE).push(postfix), 0);

    //check if the value is in range
    if (!inNumericalTypeRange(minVal))
        state.output.error("Min Gene Value out of range for data type " + i_prototype.getClass().getName()
            ,
            base.push(P_MINGENE).push(postfix),
            base.push(P_MINGENE).push(postfix));
    else minGene[index] = minVal;

    if (dynamicInitialSize)
        state.output.warnOnce("Using dynamic initial sizing, but per-gene or per-segment min-gene
            declarations. This is probably wrong. You probably want to use global min/max declarations."
            ,
            base.push(P_MINGENE).push(postfix),
            base.push(P_MINGENE).push(postfix));
}

if (minValExists)
{
    long maxVal = state.parameters.getLongWithDefault(base.push(P_MAXGENE).push(postfix), def.push(
        P_MAXGENE).push(postfix), 0);

    //check if the value is in range
    if (!inNumericalTypeRange(maxVal))
        state.output.error("Max Gene Value out of range for data type " + i_prototype.getClass().getName()
            ,
            base.push(P_MAXGENE).push(postfix),
            base.push(P_MAXGENE).push(postfix));
    else maxGene[index] = maxVal;

    if (dynamicInitialSize)
        state.output.warnOnce("Using dynamic initial sizing, but per-gene or per-segment max-gene
            declarations. This is probably wrong. You probably want to use global min/max declarations."
            ,
            base.push(P_MAXGENE).push(postfix),
            base.push(P_MAXGENE).push(postfix));
}

/// MUTATION

String mtype = state.parameters.getStringWithDefault(base.push(P_MUTATIONTYPE).push(postfix), def.push(
    P_MUTATIONTYPE).push(postfix), null);
int mutType = -1;
if (mtype == null) { } // we're cool
else if (mtype.equalsIgnoreCase(V_RESET_MUTATION))
    mutType = mutationType[index] = C_RESET_MUTATION;
else if (mtype.equalsIgnoreCase(V_RANDOM_WALK_MUTATION))
{
    mutType = mutationType[index] = C_RANDOM_WALK_MUTATION;
    state.output.warnOnce("Integer Random Walk Mutation used in IntegerVectorSpecies. Be advised that
        during initialization these genes will only be set to integer values.");
}
else
    state.output.error("IntegerVectorSpecies given a bad mutation type: " + mtype,
        base.push(P_MUTATIONTYPE).push(postfix), def.push(P_MUTATIONTYPE).push(postfix));

if (mutType == C_RANDOM_WALK_MUTATION)
{
    if (state.parameters.exists(base.push(P_RANDOM_WALK_PROBABILITY).push(postfix), def.push(
        P_RANDOM_WALK_PROBABILITY).push(postfix)))
    {
        randomWalkProbability[index] = state.parameters.getDoubleWithMax(base.push(
            P_RANDOM_WALK_PROBABILITY).push(postfix), def.push(P_RANDOM_WALK_PROBABILITY).push(postfix),
            0.0, 1.0);
        if (randomWalkProbability[index] <= 0)

```

```

        state.output.error("If it's going to use random walk mutation as a per-gene or per-segment
            type, IntegerVectorSpecies must a random walk mutation probability between 0.0 and 1.0."
            ,
            base.push(P_RANDOM_WALK_PROBABILITY).push(postfix), def.push(P_RANDOM_WALK_PROBABILITY).
                push(postfix));
    }
    else
        state.output.error("If IntegerVectorSpecies is going to use polynomial mutation as a per-gene or
            per-segment type, either the global or per-gene/per-segment random walk mutation probability
            must be defined.",
            base.push(P_RANDOM_WALK_PROBABILITY).push(postfix), def.push(P_RANDOM_WALK_PROBABILITY).push(
                postfix));

    if (state.parameters.exists(base.push(P_MUTATION_BOUNDED).push(postfix), def.push(P_MUTATION_BOUNDED).
        push(postfix)))
    {
        mutationIsBounded[index] = state.parameters.getBoolean(base.push(P_MUTATION_BOUNDED).push(postfix)
            , def.push(P_MUTATION_BOUNDED).push(postfix), true);
    }
    else if (!mutationIsBoundedDefined)
        state.output.fatal("If IntegerVectorSpecies is going to use gaussian, polynomial, or integer
            random walk mutation as a per-gene or per-segment type, the mutation bounding must be
            defined.",
            base.push(P_MUTATION_BOUNDED).push(postfix), def.push(P_MUTATION_BOUNDED).push(postfix));
    }
}

}

}

}



---


/*
 * Copyright 2006 by Sean Luke
 * Licensed under the Academic Free License version 3.0
 * See the file "LICENSE" for more information
 */

package ec.util;
import java.io.Serializable;

/*
 * Parameter.java
 * Created: Sat Aug 7 12:06:49 1999
 */

/**
 *
 * <p>A Parameter is an object which the ParameterDatabase class
 * uses as a key to associate with strings, forming a key-value pair.
 * Parameters are designed to be hierarchical in nature, consisting
 * of "path items" separated by a path separator.
 * Parameters are created either from a single path item, from an array
 * of path items, or both. For example, a parameter with the path
 * foo.bar.baz might be created from
 * <tt>new Parameter(new String[] {"foo","bar","baz"})</tt>
 *
 * <p>Parameters are not mutable -- but once a parameter is created, path
 * items may be pushed and popped from it, forming a new parameter.
 * For example, if a parameter p consists of the path foo.bar.baz,
 * p.pop() results in a new parameter whose path is foo.bar
 * This pushing and popping isn't cheap, so be sparing.
 *
 * <p>Because this system internally uses "." as its path separator, you should
 * not use that character in parts of the path that you provide; however
 * if you need some other path separator, you can change the delimiter in
 * the code trivially.
 * In fact, you can create a new Parameter with a path foo.bar.baz simply
 * by calling <tt>new Parameter("foo.bar.baz")</tt> but you'd better know
 * what you're doing.
 *
 * <p>Additionally, parameters must not contain "#", "=", non-ascii values,
 * or whitespace. Yes, a parameter path item may be empty.
 *
 * @author Sean Luke
 * @version 1.0
 */

public class Parameter implements Serializable

```

```

{
public String param;
public static final char delimiter = '.';

/** Creates a new parameter by joining the path items in s into a single path. */
public Parameter(String[] s) throws ec.util.BadParameterException
{
    if (s.length==0)
        throw new BadParameterException("Parameter created with length 0");
    for (int x=0;x<s.length;x++)
    {
        if (s[x]==null)
            throw new BadParameterException("Parameter created with null string");
        if ( x == 0) param = s[x];
        else param += ( delimiter + s[x] );
    }
}

/** Creates a new parameter from the single path item in s. */
public Parameter (String s) throws BadParameterException
{
    if (s==null)
        throw new BadParameterException("Parameter created with null string");
    param = s;
}

/** Creates a new parameter from the path item in s, plus the path items in s2. s2 may be null or empty, but
    not s */
public Parameter(String s, String[] s2)
{
    if (s==null)
        throw new BadParameterException("Parameter created with null string");
    param = s;
    for (int x=0;x<s2.length;x++)
    {
        if (s2[x]==null)
            throw new BadParameterException("Parameter created with null string");
        else param += ( delimiter + s2[x] );
    }
}

/** Returns a new parameter with s added to the end of the current path items.
    If s is empty, nothing is pushed on (and no delimiter is added to the end). */
public Parameter push(String s)
{
    if (s==null)
        throw new BadParameterException("Parameter pushed with null string");
    if (s.equals("")) return new Parameter(param);
    return new Parameter ( param + delimiter + s );
}

/** Returns a new parameter with the path items in s added to the end of the current path items. */
public Parameter push(String[] s)
{
    return new Parameter(param,s);
}

/** Returns a new parameter with one path item popped off the end. If this would result in a parameter with
    an empty collection of path items, null is returned. */
public Parameter pop()
{
    int x = param.lastIndexOf(delimiter);
    if (x==-1) // there's nothing left.
        return null;
    else return new Parameter(param.substring(0,x));
}

/** Returns a new parameter with n path items popped off the end. If this would result in a parameter with an
    empty collection of path items, null is returned. */
public Parameter popn(int n)
{
    String s = param;

    for (int y=0;y<n;y++)
    {
        int x = s.lastIndexOf(delimiter);
        if (x==-1) // there's nothing left
            return null;
        else s = param.substring(0,x);
    }
}

```



```

    }
    return new Parameter(s);
}

/** Returns the path item at the far end of the parameter. */
public String top ()
{
    int x = param.lastIndexOf(delimiter);
    if (x== -1) return param;
    else return param.substring(x+1);
}

public String toString()
{
    return param;
}

public boolean equals(Object obj)
{
    if (!(obj instanceof Parameter) || obj == null) {
        return false;
    } else {
        Parameter other = (Parameter)obj;
        return this.param.equals(other.param);
    }
}
}

package ec.util;

public interface ParameterDatabaseInf {

    boolean exists(Parameter base, Parameter def);

    int getInt(Parameter base, Parameter def, int min);

    int getIntWithDefault(Parameter base, Parameter def, int defaultInt);

    boolean getBoolean(Parameter base, Parameter def, boolean defaultBool);

    String getString(Parameter base, Parameter def);

    String getStringWithDefault(Parameter base, Parameter def, String defaultString);
}

```

## F.5 Learning Module

```

package com.montywest.marioai.learning.ec.params

abstract class DynamicSpeciesParameters {

    def minGene: Option[Long] = None
    def maxGene: Option[Long] = None
    def mutationProb: Option[Double] = None

    def numSegments: Option[Int] = None
    def segmentType: Option[String] = None

    def segmentStart(segment: Int): Option[Int] = None
    def segmentEnd(segment: Int): Option[Int] = None

    def segmentMinGene(segment: Int): Option[Int] = None
    def segmentMaxGene(segment: Int): Option[Int] = None
    def segmentMutationProb(segment: Int): Option[Int] = None

    def moduloNum: Option[Int] = None
    def moduloMinGene(congruence: Int): Option[Int] = None
    def moduloMaxGene(congruence: Int): Option[Int] = None
    def moduloMutationProb(congruence: Int): Option[Int] = None

    def minGene(index: Int): Option[Int] = None
    def maxGene(index: Int): Option[Int] = None
    def mutationProb(index: Int): Option[Double] = None
}

```

```

package com.montywest.marioai.learning.ec.params

import com.montywest.marioai.rules._

class RulesetParams extends DynamicSpeciesParameters {

  val ruleLength = Rule.TOTAL_LENGTH
  val conditionLength = Conditions.LENGTH
  val actionLength = MWAction.LENGTH

  def getIndexType(index: Int): IndexType = (index % ruleLength) match {
    case n if n < conditionLength => Condition
    case _ => Action
  }

  def runOnIndexes(indexType: IndexType, indexLimit: Int)(op: (Int,Int)=>Unit): Unit = indexType match {
    case Condition => {
      for (
        i <- 0 until indexLimit by ruleLength;
        j <- 0 until conditionLength
      ){op(i+j,j)}
    }
    case Action => {
      for(
        i <- 0 until indexLimit by ruleLength;
        j <- conditionLength until ruleLength
      ){op(i+j,j-conditionLength)}
    }
  }

  val favouredCondition = Conditions.DONT_CARE
  val favouredAction = MWAction.ACTION_FALSE

  override def minGene(index: Int): Option[Int] = getIndexType(index) match {
    case Condition => Some(Conditions.DONT_CARE)
    case Action => Some(Math.min(MWAction.ACTION_FALSE, MWAction.ACTION_TRUE))
  }

  override def maxGene(index: Int): Option[Int] = getIndexType(index) match {
    case Action => Some(Math.max(MWAction.ACTION_FALSE, MWAction.ACTION_TRUE))
    case Condition => Some(Conditions.getLimitForIndex((index % ruleLength)).toInt);
  }
}

sealed trait IndexType

case object Condition extends IndexType
case object Action extends IndexType

```

---

```

package com.montywest.marioai.learning.ec.vector

import com.montywest.marioai.learning.ec.params.DynamicSpeciesParameters
import scala.annotation.tailrec
import ec.EvolutionState
import ec.vector.VectorSpecies
import ec.util.Parameter
import ec.vector.IntegerVectorSpecies

class DynamicParameterIntegerVectorSpecies extends IntegerVectorSpecies {

  var dynamicParamsClassOpt: Option[DynamicSpeciesParameters] = None;

  override def setup(state: EvolutionState, base: Parameter): Unit = {
    val default = defaultBase

    if (dynamicParamsClassOpt.isEmpty && state.parameters.exists(base.push(DynamicParameterIntegerVectorSpecies.
      P_DYNAMIC_PARAMETER_CLASS), default.push(DynamicParameterIntegerVectorSpecies.P_DYNAMIC_PARAMETER_CLASS
    ))) {
      dynamicParamsClassOpt = {
        state.parameters.getInstanceForParameter(base.push(DynamicParameterIntegerVectorSpecies.
          P_DYNAMIC_PARAMETER_CLASS),
          default.push(DynamicParameterIntegerVectorSpecies.
            P_DYNAMIC_PARAMETER_CLASS),
          classOf[DynamicSpeciesParameters])
        match {
          case obj: DynamicSpeciesParameters => {
            Some(obj)
          }
          case _ => state.output.fatal("Dynamical parameter class wasn't a subclass of
            DynamicSpeciesParameters",

```

```

        base.push(DynamicParameterIntegerVectorSpecies.
            P_DYNAMIC_PARAMETER_CLASS),
        default.push(DynamicParameterIntegerVectorSpecies.
            P_DYNAMIC_PARAMETER_CLASS))
    }
    }
}

super.setup(state, base)
}

override def prePrototypeSetup(state: EvolutionState, base: Parameter, default: Parameter): Unit = {

    if (dynamicParamsClassOpt.isDefined) {
        val dynamicParamsClass = dynamicParamsClassOpt.get

        if (dynamicParamsClass.minGene.isDefined) {
            fill(minGene, dynamicParamsClass.minGene.get)
        }
        if (dynamicParamsClass.maxGene.isDefined) {
            fill(minGene, dynamicParamsClass.maxGene.get)
        }
        if (dynamicParamsClass.mutationProb.isDefined) {
            fill(mutationProbability, dynamicParamsClass.mutationProb.get)
        }

        if (dynamicParamsClass.moduloNum.isDefined) {
            val moduloNum = dynamicParamsClass.moduloNum.get

            if (moduloNum == 0)
                state.output.fatal(
                    "[Dynamic Params] Modulo by zero is undefined. moduloNum return must be non-zero.");

            if ((genomeSize % moduloNum) != 0)
                state.output.fatal(
                    "[Dynamic Params] moduloNum must divide genome size.");

            dynamicGenomeByModuloIndices(state, dynamicParamsClass, moduloNum);
        }

        if (dynamicParamsClass.numSegments.isDefined) {
            val numSegments = dynamicParamsClass.numSegments.get
            if (numSegments < 0)
                state.output.fatal(
                    "[Dynamic Params] Invalid number of genome segments: " + numSegments
                    + "\nIt must be a nonnegative value.");

            val segmentType = dynamicParamsClass.segmentType.getOrElse(VectorSpecies.P_SEGMENT_START)

            if (segmentType.equalsIgnoreCase(VectorSpecies.P_SEGMENT_START)) {
                this.dynamicGenomeSegmentsByStartIndices(state, dynamicParamsClass, numSegments)
            } else if (segmentType.equalsIgnoreCase(VectorSpecies.P_SEGMENT_END)) {
                this.dynamicGenomeSegmentsByEndIndices(state, dynamicParamsClass, numSegments)
            } else
                state.output.fatal(
                    "[Dynamic Params] Invalid specification of genome segment type: " + segmentType
                    + "\nThe segmentType must have the value of " + VectorSpecies.P_SEGMENT_START + " or " +
                    VectorSpecies.P_SEGMENT_END);
        }

        for (i <- 0 until genomeSize) {
            if (dynamicParamsClass.minGene(i).isDefined) {
                minGene(i) = dynamicParamsClass.minGene(i).get
            }
            if (dynamicParamsClass.maxGene(i).isDefined) {
                maxGene(i) = dynamicParamsClass.maxGene(i).get
            }
            if (dynamicParamsClass.mutationProb(i).isDefined) {
                mutationProbability(i) = dynamicParamsClass.mutationProb(i).get
            }
        }
    }

    super.prePrototypeSetup(state, base, default)
}

private def dynamicGenomeByModuloIndices(state: EvolutionState, dynamicParamsClass: DynamicSpeciesParameters,
    moduloNum: Int): Unit = {
    for (i <- 0 until moduloNum) {
        val min = dynamicParamsClass.moduloMinGene(i)
        val max = dynamicParamsClass.moduloMaxGene(i)
        val prob = dynamicParamsClass.moduloMutationProb(i)
    }
}

```

```

    if (min.isDefined || max.isDefined || prob.isDefined) {
      for (j <- i until genomeSize by moduloNum) {
        if (min.isDefined) minGene(j) = min.get
        if (max.isDefined) maxGene(j) = max.get
        if (prob.isDefined) mutationProbability(j) = prob.get
      }
    }
  }
}

private def dynamicGenomeSegmentsByStartIndices(state: EvolutionState, dynamicParamsClass:
  DynamicSpeciesParameters, numSegments: Int): Unit = {
  @tailrec
  def inner(currentSeg: Int, previousSegEnd: Int): Unit = {
    if (currentSeg < 0) return
    val segEndOpt: Option[Int] = dynamicParamsClass.segmentStart(currentSeg);
    if (segEndOpt.isDefined) {
      val segEnd: Int = segEndOpt.get

      //check if the start index is valid
      if(segEnd >= previousSegEnd || segEnd < 0)
        state.output.fatal(
          "[Dynamic Params] Invalid start index value for segment " + currentSeg + ": " + segEnd
          + "\nThe value must be smaller than " + previousSegEnd +
          " and greater than or equal to " + 0);

      //check if the index of the first segment is equal to 0
      if(currentSeg == 0 && segEnd != 0)
        state.output.fatal(
          "[Dynamic Params] Invalid start index value for the first segment " + currentSeg + ": " + segEnd
          + "\nThe value must be equal to " + 0);

      val min = dynamicParamsClass.segmentMinGene(currentSeg)
      val max = dynamicParamsClass.segmentMaxGene(currentSeg)
      val prob = dynamicParamsClass.segmentMutationProb(currentSeg)
      if (min.isDefined || max.isDefined || prob.isDefined) {
        for(j <- segEnd until previousSegEnd) {
          if (min.isDefined) minGene(j) = min.get
          if (max.isDefined) maxGene(j) = max.get
          if (prob.isDefined) mutationProbability(j) = prob.get
        }
      }

      inner(currentSeg - 1, segEnd)
    } else {
      state.output.fatal("[Dynamic Params] Genome segment " + currentSeg + " has not been defined!" +
        "\nYou must specify start indices for " + numSegments + " segment(s)")
    }
  }
  inner(numSegments - 1, genomeSize)
}

private def dynamicGenomeSegmentsByEndIndices(state: EvolutionState, dynamicParamsClass:
  DynamicSpeciesParameters, numSegments: Int): Unit = {
  @tailrec
  def inner(currentSeg: Int, previousSegEnd: Int): Unit = {
    if (currentSeg >= numSegments) return
    val segEndOpt: Option[Int] = dynamicParamsClass.segmentEnd(currentSeg);
    if (segEndOpt.isDefined) {
      val segEnd: Int = segEndOpt.get

      //check if the end index is valid
      if(segEnd <= previousSegEnd || segEnd >= genomeSize)
        state.output.fatal(
          "[Dynamic Params] Invalid end index value for segment " + currentSeg + ": " + segEnd
          + "\nThe value must be greater than " + previousSegEnd +
          " and smaller than " + genomeSize);

      //check if the index of the final segment is equal to the genomeSize
      if(currentSeg == numSegments - 1 && segEnd != (genomeSize-1))
        state.output.fatal(
          "Invalid end index value for the last segment " + currentSeg + ": " + segEnd
          + "\nThe value must be equal to the index of the last gene in the genome: " + (genomeSize-1));

      val min = dynamicParamsClass.segmentMinGene(currentSeg)
      val max = dynamicParamsClass.segmentMaxGene(currentSeg)
      val prob = dynamicParamsClass.segmentMutationProb(currentSeg)
      if (min.isDefined || max.isDefined || prob.isDefined) {
        for(j <- (previousSegEnd+1) to segEnd) {
          if (min.isDefined) minGene(j) = min.get
          if (max.isDefined) maxGene(j) = max.get
          if (prob.isDefined) mutationProbability(j) = prob.get
        }
      }
    }
  }
}

```

```

    }

    inner(currentSeg + 1, segEnd)
  } else {
    state.output.fatal("[Dynamic Params] Genome segment " + currentSeg + " has not been defined!" +
      "\nYou must specify end indices for " + numSegments + " segment(s)")
  }
}
inner(0, -1)
}
}

object DynamicParameterIntegerVectorSpecies {
  val P_DYNAMIC_PARAMETER_CLASS: String = "dynamic-param-class";
}



---



package com.montywest.marioai.learning.ec.vector

import ec.EvolutionState
import ec.util.Parameter
import com.montywest.marioai.learning.ec.params.RulesetParams
import com.montywest.marioai.learning.ec.params.Condition
import com.montywest.marioai.learning.ec.params.Action
import ec.vector.VectorSpecies

class RulesetSpecies extends DynamicParameterIntegerVectorSpecies {

  var favourMutation: Array[Boolean] = Array.fill(1)(false)
  var favourByte: Array[Byte] = Array.fill(1)(0: Byte)
  var favourProbability: Array[Double] = Array.fill(1)(0.5d)

  override def setup(state: EvolutionState, base: Parameter) {

    setupGenome(state, base);

    favourMutation = Array.fill(genomeSize)(false)
    favourProbability = Array.fill(genomeSize)(0.5)
    favourByte = Array.fill(genomeSize)(0)

    super.setup(state, base)
  }

  override def prePrototypeSetup(state: EvolutionState, base: Parameter, default: Parameter): Unit = {
    super.prePrototypeSetup(state, base, default)

    val dpc = dynamicParamsClassOpt match {
      case Some(p) => p match {
        case p: RulesetParams => {
          if (genomeSize % p.ruleLength != 0)
            state.output.fatal("Genomesize must be a multiple of rule length (" + p.ruleLength + ") ",
              base.push(VectorSpecies.P_GENOMESIZE),
              base.push(VectorSpecies.P_GENOMESIZE))
          p
        }
        case _ => {
          state.output.fatal("Dynamical parameter class must be a subclass of RulesetParams for RulesetSpecies")
          null
        }
      }
      case None => {
        state.output.fatal("Dynamical parameter class RulesetParams is required for RulesetSpecies")
        null
      }
    }

    if(state.parameters.exists(base.push(RulesetSpecies.P_CONDITION), default.push(RulesetSpecies.P_CONDITION))) {
      dpc.runOnIndexes(Condition, genomeSize){
        (x: Int, mod: Int) => {
          loadParametersForGene(state, x, base.push(RulesetSpecies.P_CONDITION), default.push(RulesetSpecies.
            P_CONDITION), "")
          loadParametersForGene(state, x, base.push(RulesetSpecies.P_CONDITION).push(""+mod), default.push(
            RulesetSpecies.P_CONDITION).push(""+mod), "")
        }
      }
    }

    if(state.parameters.exists(base.push(RulesetSpecies.P_ACTION), default.push(RulesetSpecies.P_ACTION))) {
      dpc.runOnIndexes(Action, genomeSize){

```

```

        (x: Int, mod: Int) => {
            loadParametersForGene(state, x, base.push(RulesetSpecies.P_ACTION), default.push(RulesetSpecies.
                P_ACTION), "")
            loadParametersForGene(state, x, base.push(RulesetSpecies.P_ACTION).push(""+mod), default.push(
                RulesetSpecies.P_ACTION).push(""+mod), "")
        }
    }
}

override def loadParametersForGene(state: EvolutionState, index: Int, base: Parameter, default: Parameter,
    postfix: String): Unit = {
    super.loadParametersForGene(state, index, base, default, postfix)

    // Check whether using favour mutation by checking for favour probability
    if (state.parameters.exists(base.push(RulesetSpecies.P_FAVOUR_PROBABILITY), default.push(RulesetSpecies.
        P_FAVOUR_PROBABILITY))) {
        favourMutation(index) = true;

        //Get favoured byte from params, if not found then get from dynamic params class
        val b: Byte = {
            if (state.parameters.exists(base.push(RulesetSpecies.P_FAVOUR_BYTE), default.push(RulesetSpecies.
                P_FAVOUR_BYTE))) {
                state.parameters.getInt(base.push(RulesetSpecies.P_FAVOUR_BYTE), default.push(RulesetSpecies.
                    P_FAVOUR_BYTE)).toByte
            } else {
                dynamicParamsClassOpt match {
                    case Some(p: RulesetParams) => {
                        p.getIndexType(index) match {
                            case Condition => p.favouredCondition
                            case Action => p.favouredAction
                        }
                    }
                    case _ => {
                        state.output.fatal("Dynamic paramater class RulesetParams is required for
                            RulesetSpecies")
                    }
                }
            }
        }

        //Get favour probability from params file
        val d: Double = state.parameters.getDouble(base.push(RulesetSpecies.P_FAVOUR_PROBABILITY), default.push(
            RulesetSpecies.P_FAVOUR_PROBABILITY), -1.0)
        if (d == -1.0) {
            state.output.fatal(RulesetSpecies.P_FAVOUR_BYTE + " specified without " + RulesetSpecies.
                P_FAVOUR_PROBABILITY,
                base.push(RulesetSpecies.P_FAVOUR_PROBABILITY),
                default.push(RulesetSpecies.P_FAVOUR_PROBABILITY))
        } else if (d > 1.0 || d < 0.0) {
            state.output.fatal(RulesetSpecies.P_FAVOUR_PROBABILITY + " must be between 0.0 and 1.0 inclusive",
                base.push(RulesetSpecies.P_FAVOUR_PROBABILITY),
                default.push(RulesetSpecies.P_FAVOUR_PROBABILITY))
        }
        favourByte(index) = b
        favourProbability(index) = d
    } else {
        favourMutation(index) = false
    }
}

}

object RulesetSpecies {
    val P_CONDITION = "condition"
    val P_ACTION = "action"
    val P_FAVOUR_BYTE = "favour_byte"
    val P_FAVOUR_PROBABILITY = "favour_probability"
}

```

---

```

package com.montywest.marioai.learning.ec.vector.breed

import ec.BreedingPipeline
import ec.vector.VectorDefaults
import ec.util.Parameter
import ec.Individual
import ec.EvolutionState
import com.montywest.marioai.learning.ec.vector.RulesetSpecies
import ec.vector.ByteVectorIndividual
import ec.util.MersenneTwisterFast
import scala.collection.mutable.WrappedArray

```

```

class RulesetMutationPipeline extends BreedingPipeline {

  override def defaultBase: Parameter = VectorDefaults.base.push(RulesetMutationPipeline.P_RULESET_MUTATION)
  override def numSources: Int = RulesetMutationPipeline.NUM_SOURCES

  // var mutateMu: Boolean = false
  //
  // override def setup(state: EvolutionState, base: Parameter): Unit = {
  //   val default: Parameter = defaultBase
  //   if (state.parameters.exists(base.push(RulesetMutationPipeline.P_MUTATE_MU), default.push(
  //     RulesetMutationPipeline.P_MUTATE_MU))) {
  //     mutateMu = state.parameters.getBoolean(base.push(RulesetMutationPipeline.P_MUTATE_MU), default.push(
  //       RulesetMutationPipeline.P_MUTATE_MU), false)
  //   }
  // }
  // }

  override def produce(min: Int,
    max: Int,
    start: Int,
    subpopulation: Int,
    inds: Array[Individual],
    state: EvolutionState,
    thread: Int): Int = {

    val source = sources(0)

    val n: Int = source.produce(min, max, start, subpopulation, inds, state, thread)

    // Check source of inds, clone if not a breeder
    val end = n+start
    source match {
      case bp: BreedingPipeline =>
        case _ => {
          for(i <- start until end) {
            inds(i) = inds(i).clone.asInstanceOf[Individual]
          }
        }
    }

    // CHECK INDIV AND SPECIES CLASS
    val exampleInd = inds(start)
    val rulesetSpecies: RulesetSpecies =
      (exampleInd, exampleInd.species) match {
        case (i: ByteVectorIndividual, s: RulesetSpecies) => s
        case _ => {
          state.output.fatal("RulesetMutationPipeline needs species of RulesetSpecies and individuals of
            ByteVectorIndividual")
          null
        }
      }
    state.output.exitIfErrors

    //MUTATE
    for(q <- start until end) {
      inds(q) match {
        case vecInd: ByteVectorIndividual => {
          inds(q) = this.mutateIndividual(state, thread, vecInd, rulesetSpecies)
        }
        case _ => {
          state.output.fatal("RulesetMutationPipeline needs species of RulesetSpecies and individuals of
            ByteVectorIndividual")
        }
      }
    }

    n
  }

  protected def mutateIndividual(state: EvolutionState, thread: Int, vecInd: ByteVectorIndividual, species:
    RulesetSpecies): ByteVectorIndividual = {
    for (n <- 0 until vecInd.genome.length) {
      if (state.random(thread).nextBoolean(species.mutationProbability(n))) {
        try {
          if (species.favourMutation(n)) {
            vecInd.genome(n) = getRandomByteFavouring(
              species.favourByte(n),
              species.favourProbability(n),
              species.minGene(n).toByte,
              species.maxGene(n).toByte,
              state.random(thread)
            )
          } else {
            vecInd.genome(n) = getRandomByte(

```

```

        species.minGene(n).toByte,
        species.maxGene(n).toByte,
        state.random(thread)
    )
}
} catch {
    case e: IllegalArgumentException =>
        state.output.fatal("Min is larger than max for gene " + n + ", possible overflow error.")
}
}
}
vecInd.evaluated = false
vecInd
}

protected def getRandomByteFavouring(favour: Byte, favourProbability: Double, min: Byte, max: Byte, random:
    MersenneTwisterFast): Byte = {
    if (random.nextBoolean(favourProbability)) {
        favour
    } else {
        getRandomByte(min, max, random, favour)
    }
}

protected def getRandomByte(min: Byte, max: Byte, random: MersenneTwisterFast, exceptions: Byte*): Byte = {
    var b: Byte = 0
    if (min > max) throw new IllegalArgumentException()
    else if (exceptions.length == 0) {
        do {
            b = random.nextByte
        } while (b < min || b > max);
    }
    b
    } else if (exceptions.length == 1) {
        val exception = exceptions(0);
        do {
            b = random.nextByte
        } while (b < min || b > max || b == exception);
    }
    b
    } else {
        do {
            b = random.nextByte
        } while (b < min || b > max || exceptions.contains(b));
    }
    b
}

object RulesetMutationPipeline {
    val P_RULESET_MUTATION = "ruleset-mutation";
    val NUM_SOURCES = 1;
}

```

---

```

package com.montywest.marioai.learning.ec.eval

import scala.annotation.tailrec

import com.montywest.marioai.agents.MWRulesetAgent
import com.montywest.marioai.agents.MWRulesetAgent
import com.montywest.marioai.agents.MWRulesetAgent
import com.montywest.marioai.agents.MWRulesetAgent
import com.montywest.marioai.agents.MWRulesetAgent
import com.montywest.marioai.learning.ec.params.EvaluationParamsUtil
import com.montywest.marioai.rules.KeyRight
import com.montywest.marioai.rules.KeySpeed
import com.montywest.marioai.rules.MWAction
import com.montywest.marioai.rules.Ruleset
import com.montywest.marioai.task.EvaluationTask
import com.montywest.marioai.task.MWEvaluationMultipliers
import com.montywest.marioai.task.MWEvaluationTask
import com.montywest.marioai.task.MWLevelOptions

import ch.idsia.agents.Agent
import ec.EvolutionState
import ec.Individual
import ec.Problem
import ec.simple.SimpleFitness
import ec.simple.SimpleProblemForm
import ec.util.Parameter
import ec.vector.ByteVectorIndividual

class AgentRulesetEvaluator extends Problem with SimpleProblemForm {

```



```

private var _taskSeeds: Vector[Int] = Vector(0);
def taskSeeds = _taskSeeds

private var _numberOfLevels: Int = 0
def numberOfLevels = _numberOfLevels

private var _evalMults: MWEvaluationMultipliers = MWEvaluationMultipliers.zeroEvaluationMultipliers
def evalMults = _evalMults

private var _baseLevelOptions: MWLevelOptions = MWLevelOptions.defaultOptions
def baseLevelOptions = _baseLevelOptions
private var _updateOptionsFunc: (Int, MWLevelOptions) => MWLevelOptions = MWLevelOptions.noUpdate
def updateOptionsFunc = _updateOptionsFunc

private var _fallbackAction: MWAction = MWAction()
def fallbackAction = _fallbackAction

private var task: Option[EvaluationTask] = None

override def prepareToEvaluate(state: EvolutionState, thread: Int): Unit = {
  if (task.isEmpty) {
    task = Some(
      MWEvaluationTask(numberOfLevels, evalMults, baseLevelOptions, updateOptionsFunc, false)
        .withLevelSeed(_taskSeeds(state.generation))
    )
  }
}

private def buildIndAgentName(state: EvolutionState, individual: Individual, subpop: Int, thread: Int): String
= {
  return "IND-thread:" + thread + "-gen:" + state.generation
}

override def evaluate(state: EvolutionState, individual: Individual, subpop: Int, thread: Int): Unit = {
  try {
    individual match {
      case ind: ByteVectorIndividual => {
        if (task.isDefined) {
          val evalTask = task.get
          val name = this.buildIndAgentName(state, individual, subpop, thread)
          val ruleset = Ruleset.buildFromArray(ind.genome, fallbackAction)
          val agent: Agent = MWRulesetAgent(name, ruleset)

          val iFitness = evalTask.withAgent(agent)
            .withLevelSeed(_taskSeeds(state.generation))
            .evaluate

          // state.output.message("Evalled: " + iFitness)
          ind.fitness match {
            case _: SimpleFitness => {
              ind.fitness.asInstanceOf[SimpleFitness].setFitness(state, iFitness.toDouble, false)
              ind.evaluated = true
            }
            case _ => {
              state.output.fatal("This evaluator (EvolvedAgentRulesetEvaluator) requires a individuals to have SimpleFitness")
            }
          }
        } else {
          state.output.fatal("Task was not defined when evaluating individual, implying prepareToEvaluate was not run on this instance.")
        }
      }
      case _ => {
        state.output.fatal("This evaluator (EvolvedAgentRulesetEvaluator) requires a ByteVectorIndividual")
      }
    }
  } catch {
    case e: Exception => {
      e.printStackTrace()
      state.output.fatal("Exception thrown in evaluator: " + e + " " + e.getMessage)
    }
  }
}

override def setup(state: EvolutionState, base: Parameter): Unit = {
  import AgentRulesetEvaluator._

  val default = defaultBase

  // SEEDS
  if (!state.parameters.exists(base.push(P_SEED).push(P_SEED_START), default.push(P_SEED).push(P_SEED_START)))
  {

```

```

        state.output.fatal(P_SEED + P_SEED_START + " must be defined in problem.",
            base.push(P_SEED).push(P_SEED_START), default.push(P_SEED).push(P_SEED_START))
    }

    val levelSeeds = EvaluationParamsUtil.getLevelSeeds(state.parameters, base.push(P_SEED), 0, 0, 1)

    val seedStart = levelSeeds._1
    val seedAdd = levelSeeds._2
    val seedMult = levelSeeds._3

    var prevSeed = seedStart;
    def memmedTS(g: Int): Int = {
        val ns = prevSeed + seedAdd + (g*seedMult)
        prevSeed = ns
        ns
    }

    _taskSeeds = Vector.tabulate(state.numGenerations)(memmedTS)

    //NumOfLevels
    _numberOfLevels = EvaluationParamsUtil.getNumberOfLevels(state.parameters, base.push(P_LEVEL)) match {
        case None => state.output.fatal(EvaluationParamsUtil.P_NUM_OF_LEVELS + " must be defined in params file",
            base.push(P_LEVEL)); 0
        case Some(x) => x
    }

    //EvalMults
    _evalMults = EvaluationParamsUtil.getEvaluationMultipliers(state.parameters, base.push(P_MULTS))

    //LevelOptions
    _baseLevelOptions = EvaluationParamsUtil.getBaseLevelOptions(state.parameters, base.push(P_LEVEL))
    _updateOptionsFunc = EvaluationParamsUtil.getUpdateLevelOptionsFunction(state.parameters, base.push(P_LEVEL),
        numberOfLevels)

    //Fallback
    _fallbackAction = {
        if (state.parameters.exists(base.push(P_FALLBACK), default.push(P_FALLBACK))) {
            MWAction.build(
                state.parameters.getBoolean(base.push(P_FALLBACK).push(P_ACTION_LEFT), default.push(P_FALLBACK).push(
                    P_ACTION_LEFT), false),
                state.parameters.getBoolean(base.push(P_FALLBACK).push(P_ACTION_RIGHT), default.push(P_FALLBACK).push(
                    P_ACTION_RIGHT), false),
                state.parameters.getBoolean(base.push(P_FALLBACK).push(P_ACTION_JUMP), default.push(P_FALLBACK).push(
                    P_ACTION_JUMP), false),
                state.parameters.getBoolean(base.push(P_FALLBACK).push(P_ACTION_SPEED), default.push(P_FALLBACK).push(
                    P_ACTION_SPEED), false)
            )
        } else {
            defaultFallbackAction
        }
    }

    super.setup(state, base)
}

/**
 * Should suffice to share, as all fields are internally immutable.
 */
override def clone: Object = {
    val prob = super.clone().asInstanceOf[AgentRulesetEvaluator]
    prob._taskSeeds = this.taskSeeds
    prob._numberOfLevels = this.numberOfLevels
    prob._baseLevelOptions = this.baseLevelOptions.clone
    prob._updateOptionsFunc = this.updateOptionsFunc
    prob._evalMults = this.evalMults.clone

    prob
}

}

object AgentRulesetEvaluator {

    val P_LEVEL = "level"
    val P_SEED = "seed"
    val P_SEED_START = "start"
    val P_SEED_ADD = "add"
    val P_SEED_MULT = "mult"
    val P_MULTS = "mults"
    val P_FALLBACK = "fallback-action"
    val P_ACTION_LEFT = "left"

```

```

val P_ACTION_RIGHT = "right"
val P_ACTION_JUMP = "jump"
val P_ACTION_SPEED = "speed"

private val defaultFallbackAction = MWAction(KeyRight, KeySpeed)
}

package com.montywest.marioai.learning.ec.stats

import ec.Statistics
import ec.EvolutionState
import ec.util.Parameter
import java.io.File
import java.io.IOException
import ec.Individual
import ec.vector.ByteVectorIndividual
import ec.simple.SimpleFitness
import com.montywest.marioai.learning.ec.eval.AgentRulesetEvaluator
import com.montywest.marioai.rules.Ruleset
import com.montywest.marioai.agents.MWRulesetAgent
import ch.idsia.agents.Agent
import com.montywest.marioai.agents.MWRulesetAgentIO

class RulesetEvolveStatistics extends Statistics {

  var genLog: Int = 0
  var finalLog: Int = 0
  var bestAgentFilename: Option[String] = None
  var bestAgentLimit: Int = 0
  var finalAgentFilename: Option[String] = None
  var diffAgentFilename: Option[String] = None
  var diffAgentLimit: Int = 0
  var overallBestIndividual: Option[(Int, ByteVectorIndividual)] = None
  var currentBestIndividual: Option[(Int, ByteVectorIndividual)] = None
  var biggestDiffIndividual: Option[(Int, Double, ByteVectorIndividual)] = None

  private var milliCheckpoint: Long = System.currentTimeMillis();

  override def setup(state: EvolutionState, base: Parameter): Unit = {
    super.setup(state, base)

    import RulesetEvolveStatistics._
    val genFile: File = state.parameters.getFile(base.push(P_GENFILE), null)
    if (genFile != null) {
      try {
        genLog = state.output.addLog(genFile, true)
      } catch {
        case e: IOException => state.output.fatal("An IOException occurred while making log for file " + genFile + ":\n" + e)
      }
    }

    val finalFile: File = state.parameters.getFile(base.push(P_FINALFILE), null)
    if (finalFile != null) {
      try {
        finalLog = state.output.addLog(finalFile, true)
      } catch {
        case e: IOException => state.output.fatal("An IOException occurred while making log for file " + finalFile + ":\n" + e)
      }
    }

    finalAgentFilename = {
      val str = state.parameters.getStringWithDefault(base.push(P_FINAL_AGENT_FILE), null, "")
      if (str == "") {
        None
      } else {
        Some(str)
      }
    }

    bestAgentFilename = {
      val str = state.parameters.getStringWithDefault(base.push(P_BEST_AGENT_FILE), null, "")
      if (str == "") {
        None
      } else {
        Some(str)
      }
    }

    bestAgentLimit = state.parameters.getIntWithDefault(base.push(P_BEST_AGENT_LIMIT), null, 0)
  }
}

```

```

diffAgentFilename = {
  val str = state.parameters.getStringWithDefault(base.push(P_DIFF_AGENT_FILE), null, "")
  if (str == "") {
    None
  } else {
    Some(str)
  }
}

diffAgentLimit = state.parameters.getIntWithDefault(base.push(P_DIFF_AGENT_LIMIT), null, 0)

milliCheckpoint = System.currentTimeMillis()
}

override def finalStatistics(state: EvolutionState, result: Int): Unit = {
  super.finalStatistics(state, result);

  //Write last population to finalLog
  state.population.subpops(0).printSubpopulationForHumans(state, finalLog)

  val fallback = state.evaluator.p_problem.asInstanceOf[AgentRulesetEvaluator].fallbackAction

  //Write best and last best to agent files
  if (bestAgentFilename.isDefined && overallBestIndividual.isDefined) {
    val bestRuleset: Ruleset = Ruleset.buildFromArray(overallBestIndividual.get._2.genome, fallback)
    val bestAgent: MWRulesetAgent = MWRulesetAgent("best-learnnt", bestRuleset)
    MWRulesetAgentIO.writeFile(bestAgentFilename.get, bestAgent, true)
    state.output.println("Best Agent Generation: " + overallBestIndividual.get._1.toString, finalLog)
  }

  if (diffAgentFilename.isDefined && biggestDiffIndividual.isDefined) {
    val diffRuleset: Ruleset = Ruleset.buildFromArray(biggestDiffIndividual.get._3.genome, fallback)
    val diffAgent: MWRulesetAgent = MWRulesetAgent("diff-learnnt", diffRuleset)
    MWRulesetAgentIO.writeFile(diffAgentFilename.get, diffAgent, true)
    state.output.println("Diff Agent Generation: " + biggestDiffIndividual.get._1.toString, finalLog)
  }

  //Write best and last best to agent files
  if (finalAgentFilename.isDefined && currentBestIndividual.isDefined) {
    val currentRuleset: Ruleset = Ruleset.buildFromArray(currentBestIndividual.get._2.genome, fallback)
    val currentAgent: MWRulesetAgent = MWRulesetAgent("final-learnnt", currentRuleset)
    MWRulesetAgentIO.writeFile(finalAgentFilename.get, currentAgent, true)
  }
}

override def postEvaluationStatistics(state: EvolutionState): Unit = {
  super.postEvaluationStatistics(state)

  val genNum = state.generation
  state.output.println("----- GENERATION " + genNum + " -----", genLog)

  val levelSeed: Int = state.evaluator.p_problem match {
    case eare: AgentRulesetEvaluator => eare.taskSeeds.apply(genNum)
  }
  var bestScore: Double = 0.0
  var bestInd: Option[ByteVectorIndividual] = None
  var fitnessSum: Double = 0.0

  val indivs = state.population.subpops(0).individuals
  val popSize: Int = indivs.length
  indivs.foreach { (i: Individual) => i match {
    case ind: ByteVectorIndividual => {
      ind.fitness match {
        case f: SimpleFitness => {
          fitnessSum = f.fitness + fitnessSum
          if (f.fitness > bestScore) {
            bestScore = f.fitness
            bestInd = Some(ind)
          }
        }
        case _ => state.output.fatal("This statistics class (RulesetEvolveStatistics) requires individuals with SimpleFitness")
      }
    }
  }
  case _ => state.output.fatal("This statistics class (RulesetEvolveStatistics) requires a ByteVectorIndividual")
}}
val delim = ","

val avScore = fitnessSum / popSize.toDouble
val agentStr = {

```

```

        if (bestInd.isDefined) {
            val genome = bestInd.get.genome
            val sb = StringBuilder.newBuilder
            sb.append(genome(0))
            for(i <- 1 until genome.length) sb.append(delim + genome(i).toString)
            sb.toString()
        } else "none"
    }
}

val all = "~all~ " + genNum + delim + levelSeed + delim + avScore + delim + bestScore
state.output.message(all)

val timeNowMillis = System.currentTimeMillis()
val timeTaken = ((timeNowMillis - milliCheckpoint) / 1000d).toInt
state.output.message("Time taken: " + timeTaken)
milliCheckpoint = timeNowMillis

state.output.println(all +
    "\nLevel Seed : " + levelSeed +
    "\nAverage Score : " + avScore +
    "\nBest Score : " + bestScore +
    "\nBest Agent :-" +
    "\n " + agentStr +
    "\n-----\n\n", genLog)

if (bestInd.isDefined) {
    currentBestIndividual = Some(genNum, bestInd.get)
    if (genNum >= bestAgentLimit) {
        overallBestIndividual match {
            case Some(._: Int, bvi: ByteVectorIndividual)) => bvi.fitness match {
                case f: SimpleFitness => {
                    if (bestScore >= f.fitness()) {
                        overallBestIndividual = Some(genNum, bestInd.get)
                    }
                }
            case _ => state.output.fatal("This statistics class (RulesetEvolveStatistics) requires individuals with SimpleFitness")
        }
    }
    case None => overallBestIndividual = Some(genNum, bestInd.get)
}

if (genNum >= diffAgentLimit) {
    biggestDiffIndividual match {
        case Some(._: Int, diff: Double, bvi: ByteVectorIndividual)) => {
            if ((bestScore - avScore) >= diff) {
                biggestDiffIndividual = Some(genNum, bestScore - avScore, bestInd.get)
            }
        }
        case None => biggestDiffIndividual = Some(genNum, bestScore - avScore, bestInd.get)
    }
}
}
}
}

object RulesetEvolveStatistics {

    val P_GENFILE = "gen-file"
    val P_FINALFILE = "final-file"
    val P_FINAL_AGENT_FILE = "final-agent-file"
    val P_BEST_AGENT_FILE = "best-agent-file"
    val P_BEST_AGENT_LIMIT = "best-agent-limit"
    val P_DIFF_AGENT_FILE = "diff-agent-file"
    val P_DIFF_AGENT_LIMIT = "diff-agent-limit"
}

```

## F.6 Program Entry Point

```

package com.montywest.marioai.task

import java.io.File
import com.montywest.marioai.agents.MWReactiveAgent
import com.montywest.marioai.learning.ec.params.EvaluationParamsUtil
import ec.util.ParameterDatabase
import com.montywest.marioai.agents.MWReactiveAgent
import com.montywest.marioai.agents.MWRulesetAgentIO
import ch.idsia.agents.Agent
import com.montywest.marioai.agents.MWReactiveAgent
import com.montywest.marioai.agents.MWRulesetAgent
import com.montywest.marioai.learning.ec.eval.AgentRulesetEvaluator

```

```

import java.io.IOException
import java.io.FileWriter
import com.montywest.marioai.rules.Rule
import com.montywest.marioai.agents.MWHumanAgent

object EvaluationTaskRunner {

  val LEVEL_SEED_KEY = "-seed";
  val LEVEL_SEED_DEFAULT = 52;

  val LEVEL_SEED_RUNS_KEY = "-seedRuns";
  val LEVEL_SEED_RUNS_DEFAULT = 100;

  val OUTFILE_KEY = "-outFile";
  val OUTFILE_DEFAULT = "eval-task.out";

  val NUMBER_OF_LEVELS_KEY = "-nol";
  val NUMBER_OF_LEVELS_DEFAULT = 20;

  val AGENT_FILE_KEY = "-agent"
  val AGENT_HUMAN = "human"
  val AGENT_FILE_DEFAULT = "agents/complex.agent"

  val PARAMS_KEY = "-params"
  val PARAMS_FROM_FILE = "fromFile"
  val PARAMS_COMP = "comp"
  val BASE_OPTIONS_DEFAULT = MWLevelOptions.defaultOptions
  val UPDATE_OPTIONS_DEFAULT = MWLevelOptions.noUpdate
  val EVAL_MULTS_DEFAULT = MWEvaluationMultipliers.defaultEvaluationMultipliers

  val PARAMS_FILE_KEY = "--paramsFile"
  val PARAMS_FILE_DEFAULT = "params/demo.params"

  def run(args: Array[String], vis: Boolean): Unit = {
    var argsDrop = 0;

    def getIntArg(key: String): Option[Int] = {
      args.indexOf(key) match {
        case -1 => None
        case x if x == args.length => throw new IllegalArgumentException("Key " + key + "was specied with no value")
        case x => {
          argsDrop = argsDrop + 2
          Some(args(x+1).toInt)
        }
      }
    }

    def getStringArg(key: String): Option[String] = {
      args.indexOf(key) match {
        case -1 => None
        case x if x == args.length => throw new IllegalArgumentException("Key " + key + "was specied with no value")
        case x => {
          argsDrop = argsDrop + 2
          Some(args(x+1))
        }
      }
    }

    val levelSeed: Int = getIntArg(LEVEL_SEED_KEY) match {
      case None => {
        println("Using default level seed - ")
        LEVEL_SEED_DEFAULT
      }
      case Some(x) => x
    }
    println("Level seed: " + levelSeed + "\n");

    val outfile: String = getStringArg(OUTFILE_KEY) match {
      case Some(s) => s
      case None => OUTFILE_DEFAULT;
    }

    var seedRuns: Option[(Int, Int, Int, Int)] = getIntArg(LEVEL_SEED_RUNS_KEY) match {
      case None => None
      case Some(x) => Some(x, levelSeed, 0, 1)
    }
    if (seedRuns.isDefined) println("Seed runs activated - doing " + seedRuns.get._1)

    val agent: Agent = getStringArg(AGENT_FILE_KEY) match {
      case Some(AGENT_HUMAN) => new MWHumanAgent();
    }
  }
}

```

```

        case Some(s) => MWRulesetAgentIO.fromFile(s)
        case None => MWRulesetAgentIO.fromFile(AGENT_FILE_DEFAULT)
    }
    println("Agent: " + agent.getName + "\n")

    val numberOfLevelsA = getIntArg(NUMBER_OF_LEVELS_KEY) match {
        case None => {
            println("Using default number of levels - ")
            NUMBER_OF_LEVELS_DEFAULT
        }
        case Some(x) => x
    }

    val params: (MWLevelOptions, (Int, MWLevelOptions) => MWLevelOptions, MWEvaluationMultipliers, Int) =
        getStringArg(PARAMS_KEY) match {
            case Some(PARAMS_COMP) => {
                //Throw away possible from File
                getStringArg(PARAMS_FILE_KEY)
                (MWLevelOptions.compOptions, MWLevelOptions.compUpdate(levelSeed), MWEvaluationMultipliers.
                    compEvaluationMultipliers, 512)
            }
            case Some(PARAMS_FROM_FILE) => {
                val paramsFile = getStringArg(PARAMS_FILE_KEY) match {
                    case None => {
                        println("Loading params from default file - ")
                        PARAMS_FILE_DEFAULT
                    }
                    case Some(s) => s
                }

                println("Loading params from file: " + paramsFile);
                val pd = new ParameterDatabase(new File(paramsFile).getAbsolutePath())

                val paramLevelBase = EvaluationParamsUtil.getLevelParamsBase(pd)
                if (paramLevelBase.isEmpty)
                    throw new IllegalArgumentException("Params file must contain either '"
                        + EvaluationParamsUtil.P_EVAL_BASE + "." + EvaluationParamsUtil.P_LEVEL_BASE + " = true' or '"
                        + EvaluationParamsUtil.P_LEVEL_BASE + " = true'")

                val paramMultsBase = EvaluationParamsUtil.getMultsParamsBase(pd)
                if (paramMultsBase.isEmpty)
                    throw new IllegalArgumentException("Params file must contain either '"
                        + EvaluationParamsUtil.P_EVAL_BASE + "." + EvaluationParamsUtil.P_MULTS_BASE + " = true' or '"
                        + EvaluationParamsUtil.P_MULTS_BASE + " = true'")

                val numberOfLevelsB = EvaluationParamsUtil.getNumberOfLevels(pd, paramLevelBase.get) match {
                    case None => numberOfLevelsA
                    case Some(x) => {
                        println("Overriding number of levels from param file - ")
                        x
                    }
                }

                val seedParamBase = EvaluationParamsUtil.getSeedParamBase(pd);
                if (seedParamBase.isDefined && seedRuns.isDefined) {
                    val xSeed = EvaluationParamsUtil.getLevelSeeds(pd, seedParamBase.get, levelSeed, 0, 1)
                    seedRuns = Some((seedRuns.get._1, xSeed._1, xSeed._2, xSeed._3))
                }

                ( EvaluationParamsUtil.getBaseLevelOptions(pd, paramLevelBase.get),
                  EvaluationParamsUtil.getUpdateLevelOptionsFunction(pd, paramLevelBase.get, numberOfLevelsB),
                  EvaluationParamsUtil.getEvaluationMultipliers(pd, paramMultsBase.get),
                  numberOfLevelsB)
            }
        }

    case _ => {
        (BASE_OPTIONS_DEFAULT, UPDATE_OPTIONS_DEFAULT, EVAL_MULTS_DEFAULT, numberOfLevelsA)
    }
}

val numberOfLevels: Int = params._4
// println("Number of levels: " + numberOfLevels + "\n")

val baseLevelOptions = params._1
// println(baseLevelOptions.toString())

val updateOptionsFn = params._2
// var opt = baseLevelOptions.clone
// for(i <- 0 until numberOfLevels) {
//     opt = updateOptionsFn(i, opt)
//     println(" " + i + " iter " + opt.toString())
//     println
// }

```

```

    val evals = params._3
    // println(evals.toString)

println("Running...")
if (seedRuns.isDefined) {
    val task = MWEvaluationTask(numberOfLevels, evals, baseLevelOptions, updateOptionsFn, vis, args.drop(
        argsDrop), false)
        .withLevelSeed(levelSeed).withAgent(agent)

    var fitnessSum = 0
    var bestFitness = 0
    var worstFitness = 0

    var prevSeed = seedRuns.get._2;
    def memmedTS(g: Int): Int = {
        val ns = prevSeed + seedRuns.get._3 + (g*seedRuns.get._4)
        prevSeed = ns
        ns
    }
    var writerOpt: Option[FileWriter] = None;
    try {
        writerOpt = Some(new FileWriter(outfile))
        val writer = writerOpt.get

        for (i <- 0 until seedRuns.get._1) {

            //RUN
            val seed = memmedTS(i)
            agent match {case a: MWRulesetAgent => a.resetRuleUsage; case _ =>}
            val fit = task.withLevelSeed(seed).evaluate

            fitnessSum = fitnessSum + fit
            if (fit > bestFitness) {
                bestFitness = fit
            }
            if (fit < worstFitness || worstFitness == 0) {
                worstFitness = fit
            }

            val all = "~all~ " + i + ", " + seed + ", " + fit
            println("Evalled: " + fit + " - LS: " + seed)
            writer.append(all + "\n")
        }
        val avg = fitnessSum.toDouble/seedRuns.get._1.toDouble
        println("\n\nAverage: " + avg)
        writer.append("\n\nAverage: " + avg)
        println("\nBest: " + bestFitness)
        writer.append("\nBest: " + bestFitness)
        println("\nWorst: " + worstFitness)
        writer.append("\nWorst: " + worstFitness)

        writer.flush()
    } catch {
        case e: IOException => throw new IllegalArgumentException("File inaccessible or is a folder, or error on
            flush", e)
    } finally {
        if (writerOpt.isDefined) writerOpt.get.close
    }
} else {
    val task = MWEvaluationTask(numberOfLevels, evals, baseLevelOptions, updateOptionsFn, vis, args.drop(
        argsDrop), true)
        .withLevelSeed(levelSeed).withAgent(agent)

    println
    println("Stats:")
    println
    println("Agent " + agent.getName + "'s ruleset :-")
    agent match {case a: MWRulesetAgent => println(a.ruleset); case _ =>}
    task.evaluate
    println
    agent match {case a: MWRulesetAgent => {
        for (i <- -1 until a.ruleset.length) {
            a.ruleset.ruleUsage.get(i) match {
                case None =>
                case Some(n) => println("Rule " + i + " used " + n + " times.")
            }
        }
    }; case _ =>}
    println
}

```



```

println(task.getStatistics());

var writerOpt: Option[FileWriter] = None;
try {
  writerOpt = Some(new FileWriter(outfile))
  val writer = writerOpt.get

  writer.append(task.getStatistics() + "\n\n")

  writer.append(
    task.levelScores.mkString("\n")
  )

  writer.flush()
} catch {
  case e: IOException => throw new IllegalArgumentException("File inaccessible or is a folder, or error on
flush", e)
} finally {
  if (writerOpt.isDefined) writerOpt.get.close
}
}
}
}

```

---

```

package com.montywest.marioai

import com.montywest.marioai.task.EvaluationTaskRunner
import ec.Evolve

object Main extends App {

  val MODE_KEY = "-mode"
  val LEARN = "learn"
  val WATCH = "watch"
  val EVAL = "eval"

  if (args.length < 2 ||
    args(0) != MODE_KEY ||
    (args(1) != LEARN && args(1) != WATCH && args(1) != EVAL)) {
    throw new IllegalArgumentException("First two arguments must be '" + MODE_KEY + " {learn|play|eval}'")
  }

  args(1) match {
    case LEARN => {
      println("Running learning...")
      Evolve.main(args.drop(2))
    }
    case WATCH => {
      println("Running watch...")
      EvaluationTaskRunner.run(args.drop(2), true)
    }
    case EVAL => {
      println("Running evaluation...")
      EvaluationTaskRunner.run(args.drop(2), false)
    }
  }
}

```

## References

- [1] Chad Birch, *Understanding Pac-Man Ghost Behaviour*, <http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior>, 2010.
- [2] Alex J. Champandard, *The AI From Half-Life's SDK in Retrospective*, <http://aigamedev.com/open/article/halflife-sdk/>, 2008.
- [3] Tommy Thompson, *Facing Your Fear*, <http://t2thompson.com/2014/03/02/facing-your-fear/>, 2014.
- [4] Damian Isla, *GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI*, [http://www.gamasutra.com/view/feature/130663/gdc\\_2005\\_proceeding\\_handling\\_.php](http://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php), 2005.
- [5] Siyuan Xu, *History of AI design in video games and its development in RTS games*, Department of Interactive Media & Game development, Worcester Polytechnic Institute, USA, [https://sites.google.com/site/myangelcafe/articles/history\\_ai](https://sites.google.com/site/myangelcafe/articles/history_ai).
- [6] Alex J. Champandard, *Monte-Carlo Tree Search in TOTAL WAR: Rome II Campaign AI*, <http://aigamedev.com/open/coverage/mcts-rome-ii/>, 2014.
- [7] Georgios N. Yannakakis, Pieter Spronck, Daniele Loiacono and Elisabeth Andre, *Player Modelling*, [http://yannakakis.net/wp-content/uploads/2013/08/pm\\_submitted\\_final.pdf](http://yannakakis.net/wp-content/uploads/2013/08/pm_submitted_final.pdf), 2013.
- [8] Matt Bertz, *The Technology Behind The Elder Scrolls V: Skyrim*, [http://www.gameinformer.com/games/the\\_elder Scrolls\\_v\\_skyrim/b/xbox360/archive/2011/01/17/the-technology-behind-elder-scrolls-v-skyrim.aspx](http://www.gameinformer.com/games/the_elder Scrolls_v_skyrim/b/xbox360/archive/2011/01/17/the-technology-behind-elder-scrolls-v-skyrim.aspx), 2011.
- [9] *The Berkeley Overmind Project*, University of Berkeley, California. <http://overmind.cs.berkeley.edu/>.
- [10] Simon M. Lucas, *Cellz: A simple dynamical game for testing evolutionary algorithms*, Department of Computer Science, University of Essex, Colchester, Essex, UK, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.5068&rep=rep1&type=pdf>.
- [11] G. N. Yannakakis and J. Togelius, *A Panorama of Artificial and Computational Intelligence in Games*, IEEE Transactions on Computational Intelligence and AI in Games, [http://yannakakis.net/wp-content/uploads/2014/07/panorama\\_submitted.pdf](http://yannakakis.net/wp-content/uploads/2014/07/panorama_submitted.pdf), 2014.

- [12] Julian Togelius, Noor Shaker, Sergey Karakovskiy and Georgios N. Yannakakis, *The Mario AI Championship 2009-2012*, AI Magazine 34 (3), pp. 89-92, <http://noorshaker.com/docs/theMarioAI.pdf>, 2013.
- [13] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction* The MIT Press, Cambridge, Massachusetts, London, England, Available: <http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html>, 1998.
- [14] Gary B. Fogel et al., *Evolutionary Programming* Scholarpedia, 6(4):1818., [http://www.scholarpedia.org/article/Evolutionary\\_programming](http://www.scholarpedia.org/article/Evolutionary_programming), 2011.
- [15] Stuart J. Russell, Peter Norvig, *Artificial Intelligence: A Modern Approach (3rd ed.)*, Upper Saddle River, New Jersey, 2009.
- [16] Stuart J. Russell, Peter Norvig, *Artificial Intelligence: A Modern Approach (1st ed.)*, Upper Saddle River, New Jersey, 1995.
- [17] Anoop Gupta, Charles Forgy, Allen Newell, and Robert Wedig, *Parallel Algorithms and Architectures for Rule-Based Systems*, Carnegie-Mellon University Pittsburgh, Pennsylvania, ISCA '86 Proceedings of the 13th annual international symposium on Computer architecture, pp. 28-37, 1986.
- [18] Melanie Mitchell, *An Introduction to Genetic Algorithms*, Cambridge, MA: MIT Press, 1996.
- [19] Hans-Georg Beyer and Hans-Paul Schwefel, *Evolution Strategies: A Comprehensive Introduction*, Natural Computing 1: 3?52, 2002.
- [20] S. Singh, D. Litman, M. Kearns and M. Walker, *Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJ-Fun System*, Journal of Artificial Intelligence Research (JAIR), Volume 16, pages 105-133, 2002, <http://web.eecs.umich.edu/~baveja/Papers/RLDSjair.pdf>.
- [21] Alex J. Champandard, *Making Designers Obsolete? Evolution in Game Design*, <http://aigamedev.com/open/interview/evolution-in-cityconquest/>, 2012.
- [22] James Wexler, *A look at the smarts behind Lionhead Studio's 'Black and White' and where it can and will go in the future*, University of Rochester, Rochester, NY 14627, <http://www.cs.rochester.edu/~brown/242/assts/termprojs/games.pdf>, 2002.
- [23] Thore Graepal (Ralf Herbrich, Mykel Kockenderfer, David Stern, Phil Trelford), *Learning to Play: Machine Learning in*

- Games*, Applied Games Group, Microsoft Research Cambridge, [http://www.admin.cam.ac.uk/offices/research/documents/local/events/downloads/tm/06\\_ThoreGraepel.pdf](http://www.admin.cam.ac.uk/offices/research/documents/local/events/downloads/tm/06_ThoreGraepel.pdf).
- [24] *Drivatar<sup>TM</sup> in Forza Motorsport*, <http://research.microsoft.com/en-us/projects/drivatar/forza.aspx>.
  - [25] Sergey Karakovskiy and Julian Togelius, *Mario AI Benchmark and Competitions*, <http://julian.togelius.com/Karakovskiy2012The.pdf>, 2012.
  - [26] Julian Togelius, Sergey Karakovskiy and Robin Baumgarten, *The 2009 Mario AI Competition*, <http://julian.togelius.com/Togelius2010The.pdf>, 2010.
  - [27] Julian Togelius, *How to run a successful game-based AI competition*, <http://julian.togelius.com/Togelius2014How.pdf>, 2014.
  - [28] *TORCS: The Open Racing Car Simulation*, <http://torcs.sourceforge.net/>.
  - [29] Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A. Pelta, Martin V. Butz, Thies D. Lnneker, Luigi Cardamone, Diego Perez, Yago Sez, Mike Preuss, and Jan Quadflieg, *The 2009 Simulated Car Racing Championship*, IEEE Transactions on Computational Intelligence and AI in Games, VOL. 2, NO. 2, 2010.
  - [30] *The 2K BotPrize*, <http://botprize.org/>
  - [31] Michael Buro, David Churchill, *Real-Time Strategy Game Competitions*, Association for the Advancement of Artificial Intelligence, AI Magazine, pp. 106-108, <https://skatgame.net/mburo/ps/aaai-competition-report-2012.pdf>, 2012.
  - [32] *Infinite Mario Bros.*, Created by Markus Perrson, <http://www.pcmariogames.com/infinite-mario.php>.
  - [33] H. Handa, *Dimensionality reduction of scene and enemy information in mario*, Proceedings of the IEEE Congress on Evolutionary Computation, 2011.
  - [34] S. Ross and J. A. Bagnell, *Efficient reductions for imitation learning*, International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.
  - [35] Slawomir Bojarski and Clare Bates Congdon, *REALM: A Rule-Based Evolutionary Computation Agent that Learns to Play Mario*, 2010 IEEE Conference on Computational Intelligence and Games (CIG '10) pp. 83-90.

- [36] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon, *Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution*, Proceedings of EvoApps, 2010, pp. 123-132.
- [37] E. R. Speed, *Evolving a mario agent using cuckoo search and softmax heuristics*, Proceedings of the IEEE Consumer Electronics Society's Games Innovations Conference (ICE-GIC), 2010, pp. 1-7.
- [38] Thomas Willer Sandberg, *Evolutionary Multi-Agent Potential Field based AI approach for SSC scenarios in RTS games*, University of Copenhagen, 2011.
- [39] *ECJ: A Java-based Evolutionary Computation Research System*, <https://cs.gmu.edu/~eclab/projects/ecj/>.
- [40] Sean Luke, *The ECJ Owner's Manual, v23*, <https://cs.gmu.edu/~eclab/projects/ecj/docs/manual/manual.pdf> George Mason University, 2015.
- [41] Sean Luke, *ECJ Tutorial 3: Build a Floating-Point Evolution Strategies Problem*, <https://cs.gmu.edu/~eclab/projects/ecj/docs/tutorials/tutorial3/index.html> George Mason University.
- [42] *JGAP: Java Genetic Algorithms Package*, <http://jgap.sourceforge.net/>.
- [43] Robin Baumgarten, *A\* Mario Agent*, <https://github.com/jumoel/mario-astar-robinbaumgarten>.
- [44] Ryan Small, *Agent Smith: a Real-Time Game-Playing Agent for Interactive Dynamic Games*, GECCO '08, July 12-16, 2008, Atlanta, Georgia, USA. <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2008/docs/p1839.pdf>.
- [45] Hartmut Pohlheim, *GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB Documentation*, version 3.80, 2006, <http://www.geatbx.com/docu/index.html>.
- [46] Fred Glover and Kenneth Sorensen, *Metaheuristics*, Scholarpedia, 10(4):6532., <http://www.scholarpedia.org/article/Metaheuristics>, 2015.