BM Paper

AI applied to a simple 2D environment

Principle Topic: change

Michael Gerber, Nicolas Ganz
Under the supervision of Dr. Floeder *BIA 09 7b - BMS Zürich*

Date of submission: 20.11.2012

Contents

1 Abstract

Here is some sample text to show the initial in the introductory paragraph of this template article. The color and line height of the initial can be modified in the preamble of this document.

2 Artificial Intelligence

2.1 What is AI?

This question has about as many answers as there are people studying artificial intelligence. Without getting too philosophical one could say that intelligence is defined by the efficiency of patterns used to solve problems in a complex environment and the ability to learn or generate new patterns by observing or testing.

Man-made programs to solve such problems have been around for decades and as algorithm efficiency and computational power increases they get ever more powerful.

An AI usually is directed towards one goal. This could mean solving a mathematical challenge most efficiently, finding patterns in huge amounts of data to categorize new sets of data or even to find smarter ways of learning.

2.2 History of the Field

The idea of artificial creatures acting and thinking like humans has been documented as far back as ancient Egypt. But it wasn't until the year 1955 that John McCarthy used the term artificial intelligence and defined it as "the science and engineering of making intelligent machines". Only a year later AI research became a scientific field at the campus of Dartmouth College. During the 1960s the Department of Defense was heavily funding AI research and the field was taught around the world. For the next decades research continued with big advances in problem solving but few real world applications. In 1997 the field drew a lot of public attention after IBM's chess computer Deep Blue famously defeated the reigning world champion Garry Kasparov. In the 21st century AI has found many applications in fields like medical diagnostics, data mining or household appliances like the "Roomba".

2.3 Technological Singularity

2.4 Real World Applications

Artificial Intelligence Technology

2.4.1 Amazon Recommendations

Everyone knows the "People who bought X also bought Y"-kind of recommendations online shops provide you with. Sometimes they are spot on and other times they are as far off as they can be. The idea of course is to find patterns in customer interest which can be used to offer people things they are likely to buy thereby increasing profit.

So how do they work? It is all about finding similarities in data. In the case of products you have attributes like cost, size, style, manufacturer, etc. that let classify the product. Yet taken for themselves they are close to useless because two products that are similar in size and cost might not have anything in common at all. It's even worse. Imagine two lamps that have almost the same price, the same size and the same style. What's the probability that someone who bought the one lamp would also want to buy the other? Close to zero. Now why is that? For one thing it's unlikely that someone needs two lamps. It's even less likely that they would need two almost identical lamps. So apart from flat stats about a product one needs to take into account the nature of the product to make good recommendations. An example of such would be matching light bulbs for the lamp. This tells us two things. First we need to categorize a product. Not only in it's use (e.g. lamp, chair, table, clock) but also in what frequency demand for it arises. So for example one would like to buy a new shirt every month whereas a new computer is only required every few years. The second thing is that there are certain items which depend on other items to function. A coffee machine needs filters, a toy car needs batteries and a lamp needs light bulbs. Hence to make a smart suggestion one would link those items one way so that buying a laptop would yield to recommendation of a mouse but buying cat food would not recommend acquiring a new feline companion.

This probably all sounds perfectly reasonable. Every halfway decent online shop should be capable handling categories and dependencies between products. Since the assignment of such seems more or less obvious even to an intern why not just hire some of those to do it? It doesn't require an expensive AI to figure out that it would be a good idea to recommend "Harry Potter" part two to someone who bought part one. Yet if those where to only clues to go on the list of suggestions would be pretty short for most products. To make matters worse: Since those types of relations are so obvious, it's likely the customer thought of them already so they don't yield as much additional sales as "smart" recommendations would. This is where the AI algorithms come in.

Imagine what a resale company could do if they understood their customer's taste. If they knew what style of clothing she likes, whether she lives near a beach and thus is more likely to buy a new bathing suit in spring or when her mothers birthday was so they could recommend gifts for older women at that time of year. Admittedly these examples might are little extreme yet the efforts of the so called data miners definitely go in that direction. These three examples have been picked for a reason because they each depend on different types of data.

The style for example can be derived from the clothes the customer has bought in the past. From those the retailer could categorize his customers into groups that appear to like the same products and recommend things person A from this group bought to person B. Of course there is more to it than that. For example one needs to filter out noise which may be a piece of clothing a customer bought for someone else and thus does not match the pattern. Now an especially smart AI could even make a connection between such purchases and the friend or family member they where meant to. This data then could be used for a variety of further predictions about the customers buying behavior.

The fact whether someone lives near a beach (or swimming pool for that matter) can be easily determined by matching the IP address from which requests to the store website were made to the corresponding location. There is a lot more of this type of information that one can easily read from the web request like browser type, mobile phone model, referring website and by matters of recurring visits from different locations even estimates about the customer's place of work or their boyfriend's address. Now this type of

information is only as useful as one is able to make sense of it. This is the true strength of an AI because it can find patterns in data that no one has been looking for so far.

2.4.2 Web Search Algorithms

In web search it is even more important than with large selections of products to have an automated way to make precise estimations about the value of something to someone. Furthermore the search provider not only wants to establish a prediction about how likely a person is to be looking for a certain web page but also about how likely he or she is to be looking for it right now. A lot of things factor into this like previously visited websites, behavior in choosing links and probably over a thousand other data points that might have been collected about a person. Yet even if there is no previous information about a search engine user available they can still make predictions based on the location, language and outside factors. An example of this might be the following scenario:

A user sits down in a Boston coffee shop using a browser, the signature of which has been modified to prevent it from being tracked back to that certain individual, to access her search provider of choice. She types in the word "dolphins". Now the first assumptions is that she is looking for the marine mammals. Yet the night before the football team by the same name might have played which would vastly improve the relevance of last nights game results. This increase of relevance can be justified by the fact that the user has set her language preference to American English which makes it far more likely that she is interested in the score than someone with a preference for French would be. But this isn't everything that is relevant to the search. Assuming that a band called "The Dolphins" is playing this week in Boston would make the band's website and a ticket sales site a lot more relevant to this search then it would be for someone from San Francisco.

Again this example shows how a shift in relevance might be obvious if pointed out but is extremely hard to predict for a human. Artificial intelligence can look at changes in user behavior (like increased requests to "The Dolphins" band website from Boston) and smartly adjust the ranking of certain websites. Of course this is an oversimplification of the problem but it shows nicely the basic idea of using AI to find patterns in changing data.

2.4.3 DARPA Challenges

The Defense Advanced Research Projects Agency has funded prize competitions that reward advances in research fields relevant to the Department of Defense. The initial challenge was to build autonomous ground vehicles or "self-driving cars". It was called the DARPA Grand Challenge and was first held in 2004. The goal was to build an autonomous vehicle that could drive the 240 km route through the desert fastest. None of the robot vehicles that entered the contest was able to complete the challenge. The one that performed best only finished five percent of the route before getting hung up on a rock. Therefore the event was repeated in 2005.

The second time around five vehicles were able to finish the race. With the team from Stanford University finishing first.

In 2007 the DARPA Urban Challenge has been held requiring the vehicles to finish a 96 km course in an urban area while obeying traffic regulations and interacting with human drivers.

Currently Google is developing a driverless car which has already driven 300'000 miles without an accident. The project is led by Sebastian Thrun who also was the head of the Stanford team that won the 2005 DARPA Grand Challenge.

2.4.4 IBM Watson

2.5 Classification of AI problems

There are four major attributes to an AI problem. Each of them has severe impact on how one approaches to solve that given problem.

Firstly there is the question of observability. Some environments are **fully observable** while others are only **partially observable**.

Secondly there is the factor of randomness. An environment can be completely **deterministic** which means that taking an action always result in the expected state or **stochastic**. In the latter case there is chance involved

in determining the outcome of an action. A good example for this would be a game that requires one to roll dice.

Thirdly a problem might either be **discrete** or **continuous**. The former one meaning that there is a finite number of things to do and things to observe and the latter meaning that there is a factor to the problem that is infinite and has to be handled accordingly.

Fourthly the problems environment can be **benign** or **adversarial** which would mean that there are factors to it that are actively contradicting the AI's objective by taking actions that prevent it from achieving its goals. Most games are of adversarial nature because in order to succeed each player needs to take actions that make it harder for his opponents to win.

2.6 Important AI Algorithms in Detail

2.6.1 Tree Search

In artificial intelligence many problems can be represented as trees. The first node of that tree is the initial state of the problem. From that state different actions can be taken which lead to new states. A good example of this is the game of Tic Tac Toe.

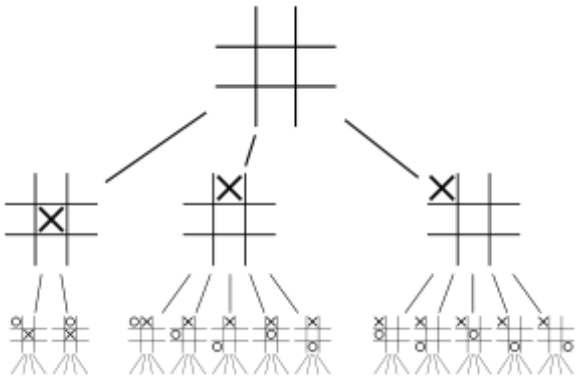

Figure 2.1: *Tree representation of Tic Tac Toe turns from Wikipedia*

There are different approaches for searching through trees each with their respective advantages and problems. Without additional knowledge about the situation the choice is between breadth-first search and depth-first search. The former one expanding always on the least explored path (broad) whereas the latter always expands the best explored (deepest) path.

At the example of Tic Tac Toe this would mean that breadth-first search would look at all the nine first turn moves, evaluating whether they result in a winning state (which of course none of them do). Next it would look at the eight resulting states the opponent's turn could have put the game in for all of those nine states. This this result in another seventy-two nodes being expanded without any of them possibly being a winning state. The first winning state could occur in turn five which means that 26'132'625 states need to be calculated and evaluated. Yet since wins in the fifth turn are only possible if the opponent is playing badly on purpose it is far more likely that the tree has to be calculated to the 7th or even ninth turn.

Now depth-first search on the other hand would just pick one first turn move and expand on one of the possibly resulting second turn moves. This process is repeated until a final state is reached, which means that there are no actions that can be taken to transition this state into another. Final states in Tic Tac Toe are wins, loses and ties. After depth-first search reached a final state it backs up to the furthest expanded node and takes another action from that state.

In a game neither of those search methods is preferable because due to the adversarial nature of the game one always has to expand every possible action the opponent could take from a state in order to determine whether that state is "good" or "bad". If there is additional information available (like rules to this game) the AI can be taught smarter tree search algorithms. For example one could teach it to see certain patterns in game states which indicate that a state is preferable or not. A depth-first search that chooses nodes to expand on an estimated value of the resulting state instead of doing so randomly is called the A* algorithm.

2.6.2 Bayes' Networks

Often the problem an AI faces is stochastic and randomness suddenly plays a big role in choosing smart actions. Using probability to make decisions is a rather simple process when all the numbers are available. It does not take artificial intelligence to figure out that an action with a 90 percent chance of success is to be preferred to one with just 40 percent. But if there are multiple events whose chance of occurring depend on each other and not all of them are observable it gets trickier.

As an example of such a problem we assume a machine that can tell us the whether there will be an earthquake today or not. But this machine is faulty and in one out of six cases it lies. Now we know from statistics 0.03 is the probability of an earthquake happening in this area. Now given that the machine tells us today is an earthquake day what is the chance of an actual earthquake happening?

To calculate this one uses **Bayes' Rule** which states this.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

This reads as: The probability of event A occurring given that we observed event B has occurred is equal to the probability of event B occurring given that event A occurred multiplied by the probability of event A occurring and normalized by the probability of event B occurring.

If event B is not known but is only dependent on event A, the rule can also be written as:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B|A) * P(A) + P(B|!A) * P(!A)}$$

Using this rule one can construct complicated networks of event that depend on each other thus making probabilities available that were previously unknown or are changed by readings of sensors.

2.6.3 Markov Decision Processes

When dealing with stochastic environments one needs to change the approach on planning. While the path of actions to a goal in a fully observable, deterministic environment can be planned in advance and can then be executed without the need to make any adjustments along the way, this is not true for stochastic environments. Because of the random nature of an actions outcome one is forced to reevaluate his priorities. Taking an action that may lead directly to the goal but also holds a small chance of resulting in an undesirable state may suddenly be much less preferable than one that requires additional steps yet holds no risk.

To account for the stochasticity of the actions one computes a value function for states that looks like this.

$$V(s) \leftarrow \left[\underset{a}{max} \gamma * \sum_{s'} P(s'|a,s) * V(s') \right] + R(s)$$

It reads as follows: The value of state **s** is the maximum over the value of all actions **a** (that can be taken from state **s**). The value of action **a** being defined by the sum over the probability of all possibly resulting states s' given that one

is in state **s** and performs action **a** multiplied by the value of this state s'. The action value is also multiplied by the discount factor γ which is set to a value smaller than one to make action sequences that take less steps than others get higher values. To this one also adds the value R of being in state **s** which may also be negative to represent the cost of getting to that state.

Using this formula the AI can calculate the best sequence of actions from any state at any time. Instead of just calculating a sequence of actions at the beginning the AI recalculates this value if possible after every step taken to maximize the chance of success.

2.6.4 Natural Language Learning

2.6.5 Particle Filters

2.6.6 Heuristics

2.6.7 Game Theory

3 Project

What do we need to create a 2 dimensional jump and run game? First of all we need a physics engine to recreate a realistic environment. Then we need a game engine to build our game on. After that we need the game itself and an environment creator to randomly generate the worlds.

Now we need an AI. But how do we connect the AI with the game? How can the AI controll the player? How can the AI see where the next block is? That's where the Application Programming Interface (API) steps in. With this connection the AI can easily read the environment and move the player. Lastly we "just" need the AI algorithms.

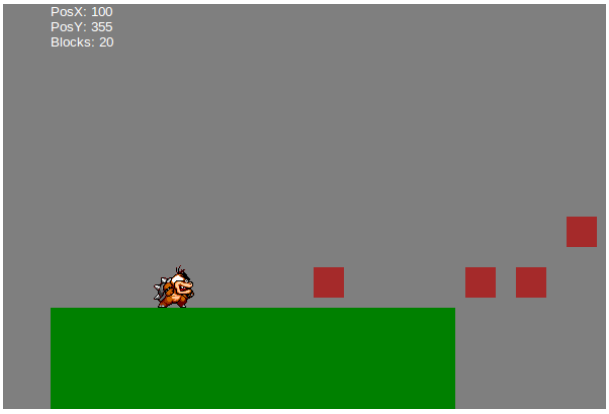

Figure 3.1: *The final game at the start point.*

3.1 Physics Engine

A physics engine is able to recreate an entire physical system like our world. They are mostly common in animated films, scientific experiments and video games. The gravity is the biggest part to cover. There are less and more accurate engines. In some the falling objects get faster or the objects can break when they fall on a solid ground.

3.1.1 Gravity Example

A block gets thrown horizontally with the force F. This block has now the force from the throw (horizontally) and from the gravity (vertically).

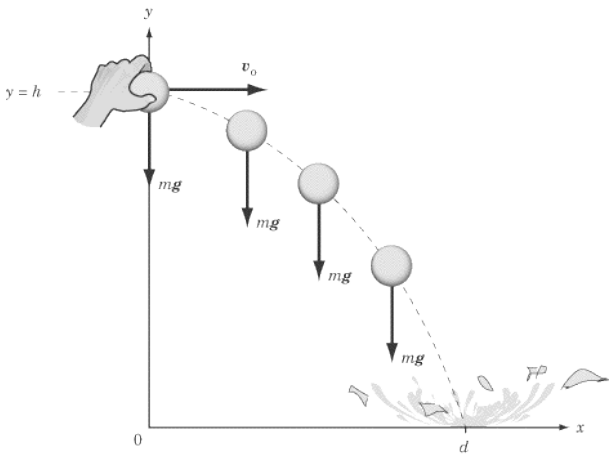

Figure 3.2: *Example for gravity.*

3.1.2 Collision Example

A block gets thrown horizontally with the force F and collides with another, steady one with the same size and weight. After the collision the first block will fly back with half the force, while the other one gets pushed away with the other half of the force.

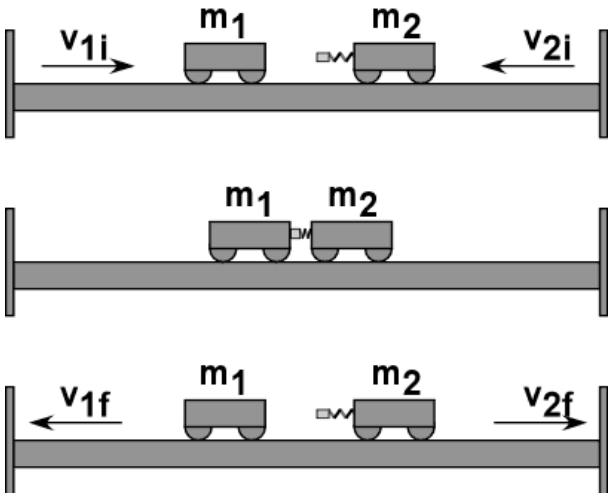

Figure 3.3: *Example for a collision.*

3.1.3 Types of Physics Engines

There are two main types of physics engines. The high-precision and the real-time engines:

high-precision This type of engine is used for exact copies of a physical system. They are mostly common for animations that must be very precise like scientific studies or animated films. It does not matter in this animations how many resources where used while rendering because you can let the machine run for many hours rendering the world and than analyze the result at the end.

To wrap it up: Those engines need much processing powers but are very precise. Quality before quantity.

real-time This type of engine is used for interactive animations like video games. The artificial physics must be calculated immediately. Unlike the high-precision engines this type of engine does not need much processing power, for it is minimized.

To wrap it up: The real-time engines do not need much processing power but are not very accurate. Quantity before quality.

3.1.4 What do we need?

collision We need to be able to check for a collision between the player and a block above the ground.

gravity We need to be able to use gravity for the player to jump and fall down.

real-time We are creating a game, so we need in the first place a fast engine.

3.1.5 Setting Boundaries

But let us stop here. Even though it is a really interesting topic, it is too extensive to go into the details. If you would like to know more, there are plenty of comprehensive studies about physics engines.

3.2 Game Engine

We want the physics engine to create a game. But how can one do this? All by himself? No, we need to find a link between the game and the physics engine. This is where the game engine has a turn. It offers the opportunity never to touch the physics engine and to make the step from a world with just moving objects to a world, controlled by a human.

3.2.1 What does a Game Engine?

- It unburdens the interaction between the human's controls like a keystroke or a mouse click and the animations that can be seen on the display.
- It eases the animations of the objects. For example when the player walks then should the character animate this movement.
- It implements the element of sound. With this feature you can easily add background music and/or sound effects.
- They are mostly specialized on 2D or 3D worlds. 3D worlds are way more complicated but they are more realistic.
- They are either built for a specific platform or they can compile the same source code for multiple platforms. For example if it supports multiple platforms you can create a game and run it on Windows, Linux, Xbox and PlayStation.
- Some engines even include a AI system to help building one. They often helps with the data logistics and sensor access.

3.2.2 Examples of Game Engines

CryENGINE A engine created by Crytek. It runs on PS3, Xbox 360 and PC. It has an advanced AI system with sensors like sight and hearing. The games Crysis, FarCry and Warface are the most famous of many.

Unreal Engine A engine created by Epic Games. It runs on PC, PS3, Xbox 360, Wii U, PSVITA, Android, iOS and Flash. The Unreal AI is also advanced and includes a special navigation mesh system to optimize the performance and memory usage. Dishonored, Gears of War, Batman: Arkham City and Mass Effect are the best examples for this engine.

Anvil Until 2006 it was known as Scimitar and is created by Ubisoft. It runs on PC, PS3, Xbox 360, Wii U and PSVITA. Prince of Persia, Assassin's Creed and Tom Clancy's Rainbow 6: Patriots are the most known examples for this engine.

IW Engine A engine created by Infinity Ward. It runs on PC, PS3, Xbox 360, Wii and Wii U. The complete Call of Duty series expect the first one is based on this engine.

Blender This is the most popular open source game engine.

3.2.3 What do we need?

We don't need a complex game engine for we are creating a simple, 2 dimensional game. It should just cover the basic necessities like:

- interaction between user interface and animations
- animations for the objects
- sound implementations
- built for PC only should be sufficient
- 2 dimensional

3.3 The Game

For it would take too much time to create an own physics and game engine, we decided to use the open source engine Crafty. It is based on the programming language JavaScript allowing the game to run in a modern web browser.

3.3.1 Goal

We want to create a game with this specifications:

- 2 dimensional
- jump and run
- blocks to jump on
- gaps in the floor to fall down
- randomly generated
- generator ensures that the game is solvable
- finish line at the end of the parcour
- die from falling down
- win from reaching the finish line
- after winning or dying: restart the game

3.3.2 Basic Composition

We used an event-driven development architecture. That means that the engine triggers events and we define what should happen.

Here is the main structure of the game in a theoretically programming language:

```
When the player hits a solid object:
  The player stops moving

When the player hits a deadly area:
  The player dies
  Restart the game

When the player hits the finish line:
  The player wins
  Restart the game
```

3.3.3 Animation Sprites

To bring life to our player we decided to use a sprite. A sprite is basically a collection of images with the same character but in different positions. You just have to decide which of them belongs to which movement.

Figure 3.4: *The sprite to animate our player.*

standing right If the player stands and looks to the right the game should use the image from row 1 the first

standing left If the player stands and looks to the left the game should use the image from row 2 the first

jumping right If the player jumps to the right the game should use the image from row 1 the last

jumping left If the player jumps to the left the game should use the image from row 2 the last

walking right If the player walks to the right the game should switch through all images from the third row

walking left If the player walks to the left the game should switch through all images from the forth row

3.3.4 Environment Creator

Our environment consists of blocks and a floor with gaps. We now need a creator that generates random environments. To make it consistent we chose that all blocks have the same size. They are in one of this three heights:

1. The height of the player. Easy to jump on.
2. Twice the height of the player. The maximal height to jump on.
3. Thrice the height of the player. Just reachable from a block from the first or second height.

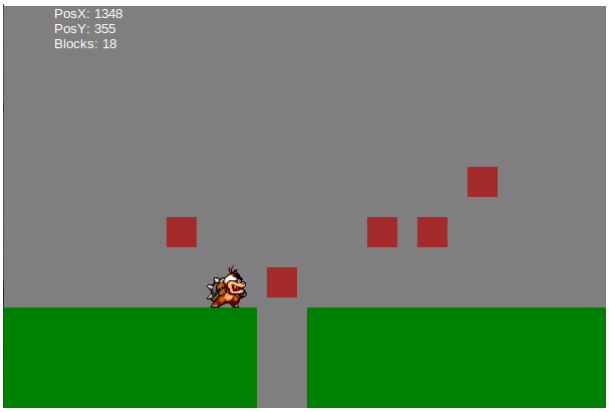

Figure 3.5: *The three possible block heights.*

The gaps in the floor exist in three different sizes:

one block These gaps are very easy to jump over.

two blocks These are a little harder, but should pose no problem.

three blocks These are the most difficult ones. To jump over them you have to stand on the very last corner of the floor or on a block.

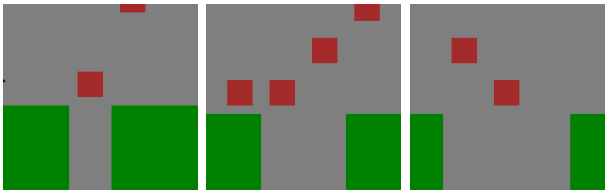

Figure 3.6: *The three possible sizes of the gaps.*

Prevent Unsolvable Environments

Basically we could just say "create some blocks and some gaps in the ground" but it would most likely get unsolvable. To prevent this we have chosen this insurances to prevent that:

- The blocks from the first or second height are no problem for the player to jump on.
- The blocks on the third height may not exist if they don't have an accessible block from the first or second height. This is to make sure that the player can jump on every block.
- The gaps may not be bigger than the player could jump.

Composition

This is the basic composition of the creator in a theoretical programming language:

```
Define the start point
Until the whole width of the field is used:
  Set a random end point
  Create a ground from the start to end point
  Get a random gap width (1, 2 or 3)
  Set the start point: end point + gap width

15 times:
  Until the x-coordinate is unique:
    Generate random x-coordinate

  Get a random height of 1 or 2
  Generate the block from x and the height

  With a probability of 3:
    Generate a next block with the height 3
```

3.4 Application Programming Interface (API)

As mentioned before we need to create an interface for the AI to control the player and read the environment. To gain full control we need actors and sensors.

For the human the actors are the keys and his fingers with which he can handle the player. The sensors are the monitor and his eyes. With this he can analyze what is happening.

Now we have to adapt those actors and sensors to our API.

3.4.1 Actors

The actors are responsible for the actions of the player. We need those available actions:

jump The AI must be able to jump to the left and the right.

walk The AI must be able to walk to the left and the right.

3.4.2 Sensors

The sensors are responsible for the measuring of the environment. We need those available sensors:

won? Did the player win?

died? Did the player die?

my position Where is the position of the player?

nearest block Where is the nearest block?

all blocks Where are all blocks? This is needed to plan a few steps further

nearest gap Where is the nearest gap?

all gaps Where are all gaps?

finish line Where is the finish line? To check if the AI is going in to the right direction.

3.4.3 Implementation

Those actors and sensors sound all very logical but how could we implement those?

For the sensors we just have to access the created elements. Of course they need to be well formed and easily accessible. For the nearest gap or block we just have to iterate through all and check which of those the closest one is.

For the actors it is more complicated. We have to include multiple callbacks and helper variables to ease the access. For example we need a callback that will be triggered when the player lands on the ground or when he have reached the desired distance.

3.5 Our Artificial Intelligence (AI)

4 Summary

5 Sources and Glossary

5.1 Sources

Physics Engines

- Physics engine - Wikipedia (http://en.wikipedia.org/wiki/Physics_engine)

Game Engines

- Game engine - Wikipedia (http://en.wikipedia.org/wiki/Game_engine)
- CryENGINE | Crytek (http://www.crytek.com/cryengine)
- Game Engine Technology by Unreal (http://www.unrealengine.com)
- Anvil (game engine) - Wikipedia (http://en.wikipedia.org/wiki/Anvil_(game_engine))
- IW engine - Wikipedia (http://en.wikipedia.org/wiki/IW_engine))

The Game

- Crafty - Javascript Game Engine (http://craftyjs.com/)

5.2 Image Sources

List of Figures