Intelligent Monsters: Downfall of the Gamer

Making game AI more interesting with predictive processing

Name
Bannink, W.R.J. (Ward)
ward.bannink@student.ru.nl

Student administration number
s4235061

Study
Artificial Intelligence, Radboud University Nijmegen

Supervisors
Kwisthout, J.H.P. (Johan)
j.kwisthout@donders.ru.nl

Otworowska, M.E. (Maria)
m.otworowska@donders.ru.nl

1. Abstract

Due to the lack of recent development in game AI, new ideas may very well reinvigorate research and development in this area. This study aims to do exactly that with predictive processing. The predictive processing account proposes that the human brain attempts to predict future states, and that it subsequently tries to minimize the error of these predictions. The aims of this study to create a game AI using predictive processing to predict player moves in a grid-based world. As this study was meant to be explorative, an informal pilot was carried out, as opposed to a formal experiment. Five participants would play the game under observance, and provide feedback when done. Multiple rounds were played against an AI utilizing predictive processing, as well as against a simple AI that simply chased the player. Feedback and impressions from the gameplay suggest that the participants preferred the predictive AI above the simple AI.

2. Preface

This bachelor thesis has been written by W.R.J. Bannink, student artificial intelligence at the Radboud University of Nijmegen. For deciding what to do me thesis on, I was offered multiple projects to participate in for my thesis, among which was the predictive processing project. As I am really interested in the development of games and artificial intelligence, the Predict Project caught my attention, because it could be a very interesting thing to use with game AI. After having clarified my intentions and preferences with my supervisor, we devised a way to combine game AI and predictive processing, the result of which written in this thesis.

Before we begin, I'd like to mention the following people who have helped me throughout the process of this project. First and foremost, my supervisors: Johan Kwisthout, and Maria Otworowska without whom I'd never have been able to complete this. Furthermore I would like to thank Robert-Jan Bannink, Maria Bannink, Job Bannink, Sven Herden, Jesse Fenneman, Maaike ter Borg, Harmen Prins, Maaike Deelstra, Daniel Stremmelaar, Auke Rozema, and Daan Vos for their support and helping me to get through.

Table of Contents

3. Introduction

3.1 Intro

For the past few years the development of artificial intelligence in games has been rather stale (Yannakakis, 2012). A lot of games nowadays are using the same kind of algorithms over and over again, with some of them making small additions to the trusted formula, or fake difficulty by letting the AI (artificial intelligence) cheat. While there have been some attempts at making an AI appear more human-like, this tends to take the form of making the AI imitate human behaviour (Wang, Subagdja, Tan, Ng, 2009). While this can be efficient and impressive, it doesn't actually make the artificial intelligence 'intelligent'.

My theory is that as things currently stand, there is room for improvement in the department of game AI, as there is a steady increase in the computing power of computers and developers are allowed to spend more resources on AI (Nareyek, 2004). Nowadays, AI tends to display rather predictable and static behaviour, whereas humans tend to be a lot more unpredictable. Which may be a reason why a lot of people prefer to play multiplayer games, as humans are simply more interesting to play against. In order to improve AI in this area I have investigated whether the following things can create more human-like behaviour:

- constructing a generative model of the player's behaviour
- making predictions according to this model
- adapting behaviour and/or the model using prediction errors

To do this, I have used a concept from a new theory in neuroscience, namely predictive processing (Kwisthout, Bekkering, & van Rooij (in press)). There is a good opportunity to make an AI that utilises concepts from predictive processing, one that is more humane than the general AI. There will have to be a balance between the performance and fun-factor, as an AI that will correctly predict everything all the time isn't any fun to play against. On the other hand, an AI that isn't predictable, seems to employ different strategies, and generally appear more human-like is received better (http://www.kurzweilai.net/ai-game-bots-more-human-like-than-half-of-human-competitors).

For this research, I have investigated the following research question: 'Can predictive processing be used to produce a more realistic artificial intelligence in games?' As previously stated, the project this thesis is part of has predictive processing as its main topic. Apart from that, I want to find a job in game development after finishing my education. These two things combined allowed me to work on something that fit within the frame of the project, as well as accumulate experience with creating a game. Both of which will be beneficial later on, as predictive processing sounds like an interesting thing to use within games. In order to get an answer to this research question I have used the game I created as a testing ground. Participants have been invited to play a number of rounds against a simple AI and the predictive processing AI and judge them on various components.

Lastly, for this part, follows the general outline of the thesis. The remainder of the introduction consists of three parts; 'Setup', 'Predictive Processing', and 'Important terms, distinctions, and formulae'. In the setup, background information is provided on the game created for this project, and on the game itself in a nutshell. Predictive processing will briefly explain the idea behind the predictive processing account. The important terms, distinctions, and formulae will provide the reader with some important terms that are used in the thesis.

After the introduction comes 'Model', where everything about the theoretic model I have used for the game is explained. This part is split up in 'Introductory world example' that makes the reader

accustomed with how predictive processing was used to create the AI. This is done by means of an exemplary, smaller, world. 'Main world model' then takes the exemplary world, and scales it up to the actual world used within the game, and shows what was done in order to create a working AI utilizing predictive processing.

In 'Methods', more specific information is given on the game itself. It is split up in three parts. 'Information trace of the AI' shows an exemplary game in progress and offers insight into the AI. Like what the AI knows, and what its intended path is. 'Program mechanisms' outlines the classes used within the code, briefly explains what task they had within the game, and mentions some of the important functions. 'Anecdotal pilot setup' contains information about the informal 'experiment' that was carried out.

'Observations' provides information about 'Impressions of the gameplay', 'Feedback and commentary', and 'Anecdotal results'. Together, these form an impression of what the game was like for the participants, and some noteworthy observations.

In 'Conclusion' I will take the gathered information and observations and draw conclusions from them. But seeing as this was a pilot instead of an actual experiment, these are not facts. Following up is 'Discussion', in which possible explanations for the observations are given.

Lastly, 'Further research' offers some idea to research that build further upon the pilot. 'References' contain the references to the sources of information used in this thesis, and in 'Appendix' the code for two important functions is to be found.

3.2 Setup

To explore whether predictive processing mechanisms can offer a valuable contribution to game AI behaviour, I have made an AI for a dungeon crawler type of game, which can be compared to the well-known Wumpus (from the game 'Hunt the Wumpus, developed by Gregory Yob).

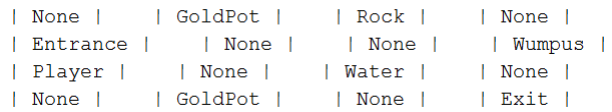

Illustration 1 A screenshot of the game in progress

A dungeon crawler typically is a game in which the player navigates through a labyrinthine environment, the dungeon, in search of gold, powerful loot, and monsters to slay. Most of the time, there isn't a lot of story involved in dungeon crawlers, resulting in straight to the point games that can still be somewhat complex due to the amount of options the game can offer. This usually takes the form of different races, classes, and items to make the player stronger. Hunt the Wumpus is similar to a dungeon crawler; the player is armed with a bow and tasked to hunt the creature known as the Wumpus. Each turn he/she can explore a different room, or fire the bow to a room. If the arrow is shot to a room containing the Wumpus, the player wins, although there could be multiple wumpi in a single cave. If he/she steps into the room where Wumpus is located, the player loses. Furthermore, some feedback is provided to the player about where the Wumpus is. Namely, the player can smell the Wumpus from two rooms away. A lot of guesswork and manually drawing out the maps was needed for this game as it was text-based.

In the case of the game for this project, the world consists of a two-dimensional grid, filled with certain features like rocks, water, and pots of gold, the Wumpus (the monster trying to eat the player) as well as an entrance and exit for the player. The game consists of a player moving within the grid, trying to gather as much gold as possible to increase the player's score, and then move to the exit, all the while avoiding, or attempting to kill, the Wumpus. It is easier to guess where the Wumpus is in this game, as there are line of sight mechanics that determines whether player and Wumpus are visible to each other.

3.3 Predictive Processing

The predictive processing account proposes that the human brain is a hierarchically organized structure that tests hypotheses (Kwisthout, Bekkering, van Rooij, 2016). It does so by making predictions about its environment, and subsequently attempts to minimize the error of these predictions (Clark, 2012). This hierarchy is organized in increasingly abstract probabilistic predictions, and the hypothesized causes that drive the predictions. At each level of the hierarchy, the predictions about the inputs are compared with the actual inputs, and possible prediction errors are minimized (Kwisthout, Bekkering, van Rooij, 2016). The generative models generate the hypotheses used within the account based on the inputs (Yoshida, Dickey, Sturt, 2013), but how these models are created and change over time hasn't been looked into much yet.

The inputs to the brain are assumed to be predicted in a hierarchical manner by the generative causal models. Of all of these inputs, the only bits that get actually processed are the yet unexplained parts, which are the prediction errors. These stem from the inherent stochastic nature of the world (Kwisthout, Bekkering, van Rooij, 2016). To make sense of what caused these errors, the brain tries to explain the prediction errors away with various mechanisms. Among these are adding additional observations (Friston, Adams, Perrinet, Breakspear, 2012), actively intervening in the world (Brown, Friston, Bestmann, 2011), updating the hypotheses (Friston, 2002), or revising the model (Friston, 2003).

3.4 Important terms, distinctions, and formulae

- Probability distribution over the actions: for each specific action the probability of the player carrying out that action

- Observed probability distribution: the probability distribution that corresponds with the observed player action. If the action is fully observable, the distribution will be deterministic.

- Predicted probability distribution: the probability distribution that is calculated by the program and is based both on the current world state, and the player strategy.

- Prediction error: the difference between the observed probability distribution and the predicted probability distribution.

- Prediction error minimalization; a method that seeks to minimize the size of the prediction error by updating or modifying the generative models.

- Kullback-Leibler divergence: a measure that is used to calculate how much two probability distributions differ from each other. In predictive processing, this is used to indicate the size of the prediction error. It is calculated with the following formula:

$$D_{KL}(Pr_{(Obs)} \,||\, Pr_{(Pred)}) = \sum_{p \in \Omega(Obs)} Pr_{Obs}(p)\, log_2 \left(\frac{Pr_{Obs}(p)}{Pr_{Pred}(p)} \right)$$

- Line of sight: the mechanism used to determine whether the player and Wumpus can see each other, abbreviated to LOS. When there is LOS, the Wumpus is able to observe player actions and use them to predict future player actions.

- Pot of gold: on object that can be located at locations within the game field. When the player walks over a pot of gold, it is picked up and the player's total gold gets increased by 10.

4. Model

This section contains information about the employed model. Before explaining things in a more detailed version, some more formal definitions of the model, and of variables and values used within the model are provided here.

There are two strategies that the AI assumes the player to utilize; gold-first, and exit-first. As together these two make up the probability distribution over the strategies, each strategy has a probability of P(Strategy = s) = x. Where s is either gold-first or exit-first, and x is the probability for that strategy to be assumed by the AI.

The actions a player can take consist of moving in one of the four cardinal directions; north, east, south, and west. As with the probability for a strategy, the probability for an action can be described as P(Action = a) = x, where a is one of the possible actions a player can take, and x is the probability assigned to that action.

More on that, these probabilities are calculated based on the assumed strategy and world state. Let's assume that we have some world state w, and some assumed strategy s. In accordance with this, the actual probability for an action would be P(Action = a | w, s) = x.

The world state itself is something like a screenshot of the world, or in the case the game field. Different world states can be created by, for example, due to player movement, and picking up pots of gold. It can be seen as a Bayesian network, with the starting state being the root node. As actions are taken the network expands, creating a new node depending on the chosen action of player and AI, and creating possible successor nodes depending on the possible actions for the player and AI.

4.1 Introductory world example

Performing all of the computations necessary to have a functioning game played out within a decently-sized world would be quite the complex task, due to all the possibilities that have to be considered when predicting a single action. As of such, the general thinking process of the AI will be illustrated by means of a smaller exemplary world, shown to the right. This world contains a player, a single pot of gold, and the exit. In this case, the AI assumes the player will utilize one of two strategies at a time: either gold-first, or exit-first. As there is no Wumpus within the world state, this leaves the player with 4 possible actions; moving either vertical or horizontal.

	Pot of Gold
Player	
	Exit

The foundation of action prediction is based on the current world state, and probabilities of employment of the strategies by the player (see Illustration 1). The employed strategy indicates the player's believed intentions and goals, whereas the world state pertains the status quo of the game. These two things combined allow the AI to make predictions. These predictions take the form of a probability for each action, which are conditionally dependent on the strategies and state of the world. Actions that are consistent with the assumed strategies thus have a high probability, while those that don't will get a low probability. There is a reason these actions are assigned a low probability instead of plain zero. This is because when a zero probability would be assigned to these actions, and the player takes them anyway, there would be infinite prediction errors. These infinite prediction errors stem from the resulting outcome of the Kullback Leibler divergence. Should the predicted probability be zero, the following would attempt to execute: '$\log_2$ (observed probability / 0)', and log(0) is minus infinity. Therefore, the actions that, according to the prediction, won't be taken will be regarded as

noise. Impossible actions will still retain their zero probability, as the player simply isn't able to execute them.

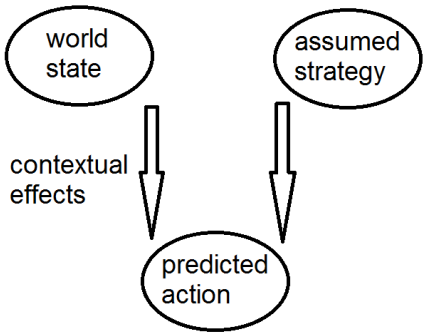

Illustration 1 A simple depiction of how a predicted action is formed; through processing of information from the assumed player strategy, and the current world state and its contextual effects.

In this case, initially the strategies are uniformly distributed. That is to say, each strategy will have a probability of 0.5, i.e. P(Strategy = GoldFirst) = 0.5, and P(Strategy = ExitFirst) = 0.5. This is done in order to reflect the AI's lack of knowledge which specific strategy is actually being pursued.
In this case, under the assumption that a gold-first strategy is being pursued, the actions 'move east' and 'move north' will have a probability of 0.49, as both bring the player equally closer to his/her assumed goal. 'Move west' is ruled out due to the constraints imposed by the game. 'Move south' will have the remaining probability of 0.02, functioning as noise to retain a predicted probability that the player performs this action. This is not to say it is impossible for this action to be carried out by the player, but to the AI this is an illogical action (as it increases the distance between player and gold), therefore these actions will be assigned some small probability to serve as noise. Largely the same will go for when the exit-first strategy is assumed, except that 'move north' will be assigned a probability of 0.02, and 'move south' will have a probability of 0.49 now.

Given that West is an impossible action in this cell, and there are two actions for both the GoldFirst and ExitFirst strategy that are equally likely (and one action in each case that is unlikely), the conditional probability table is defined as follows. Here, we implicitly condition also on the world state which in our model is fully observed:

P(North | Strat = GoldFirst)	0.49
P(East | Strat = GoldFirst)	0.49
P(South | Strat = GoldFirst)	0.02
P(South | Strat = ExitFirst)	0.49
P(East | Strat = ExitFirst)	0.49
P(North | Strat = ExitFirst)	0.02

The predicted distribution over the actions, given the uncertainty about which strategy is being employed, is computed as follows:
P(Action = x) = P(x| Strat = GoldFirst) * P(Strat = GoldFirst) + P(x | Strat = ExitFirst) * P(Strat = ExitFirst). Resulting in the next marginal probability distribution, which will function as the predicted probability distribution for now:

P(North)	0.49 * 0.5 + 0.02 * 0.5 = 0.255
P(East)	0.49 * 0.5 + 0.49 * 0.5 = 0.49
P(West)	0 * 0.5 + 0 * 0.5 = 0
P(South)	0.02 * 0.5 + 0.49 * 0.5 = 0.255

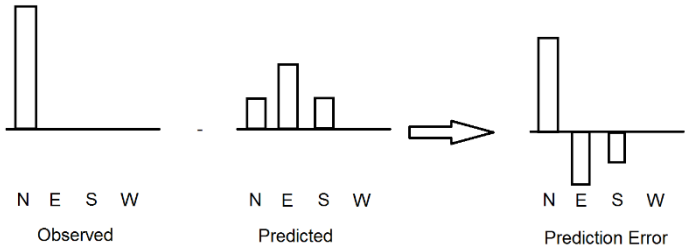

Illustration 2 The prediction error is calculated by subtracting the predicted probability distribution from the observed probability distribution.

After the player has performed an action there will be an observed probability distribution over the action variable, as well as a probability distribution with the predicted probabilities. The AI will compute the prediction error between predicted and observed actions by subtracting predicted distribution from the observed one, as shown in Illustration 2, and update its current beliefs and/or model accordingly to lower the size of the prediction error (defined as the Kullback-Leibler (KL) divergence between both probability distributions). Should, for example, the player choose to move north, this induces a prediction error. This is caused due to the discrepancy between the predicted and observed probability distributions. One way of lowering the size of the prediction error is by revising the prior probability distribution over the strategies. More precisely in this case: the probability of the gold-first strategy will increase, as this is the strategy that would best predict this action, given the context the action was taken in. Thus, the probability distribution over the Strat variable is updated such that the KL divergence between the observed probability distribution over Action and the resulting predicted distribution over Action is minimal, this happens to be the distribution where P(Strat = GoldFirst) = 1, as shown in Illustration 3 below.

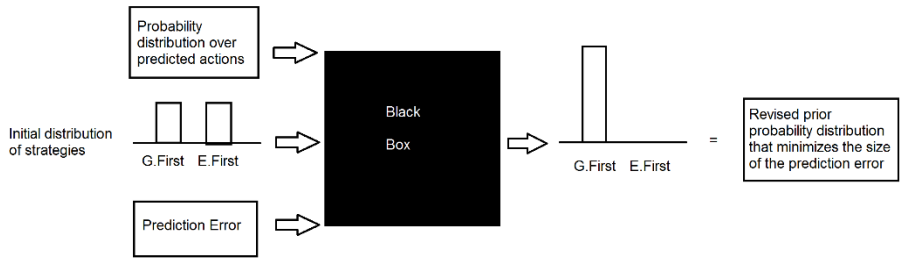

Illustration 3 Creating the revised probability distribution.

This updated distribution is then used in combination with the new state of the world to make the new predictions for the next action of the player. The new probability distribution for predicted actions for the gold-first strategy would consist of a probability of 0.99 being assigned to ‘move east’, as it is the only possible action that fits this strategy, given the new world state and beliefs. The other legal action (going South) is unlikely given this strategy), as the AI currently assumes the gold-first strategy is employed. Meanwhile, the probability of the exit-first strategy will decrease, as this action increased the distance between player and exit. All of which is shown in the following table:

P(North)	0 * 1 + 0 * 0 = 0
P(East)	0.99 * 1 + 0.5 * 0 = 0.99
P(West)	0 * 1 + 0 * 0 = 0
P(South)	0.01 * 1 + 0.5 * 0 = 0.01

On the other hand, should the player choose to move east (instead of north), there will be a smaller prediction error, as the observed action was one that had a high probability (see Illustration 4). The beliefs about the strategies remain identical to the previous state, as the action is equally consistent with both strategies. As of such, the probability of the strategies will remain the same, as the action was a legitimate move for both. Lastly, if the choice was made to move south, the beliefs revision will decrease the probability of gold-first. Increasing exit-first instead, and predicting the next action accordingly.

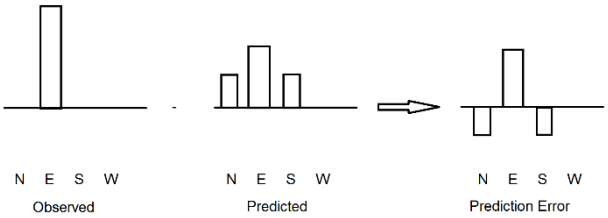

Illustration 4 Resulting prediction error when the player moves east.

4.2 Main world model

However, the actual computations are a lot more complex, because the actual game is played out within a much larger world. This combined with the improbability of the player utilizing a single strategy, as players tend to change strategies depending on the situation (Milchtaich, 1996), quickly causes things to spiral into intractability. Even when this would be represented as a probability distribution over the possible strategies, the large number of world states and actions possible in each world state cause intractability to occur. Along with which come huge probability tables that would require numerous pages to be illustrated.

Now that we've taken a look at a small example of what has been done, let's take it to the next level and see how things go about within a larger world. This 4x4 world consist of a player, some pots of gold, an entrance, exit, landmarks, and the Wumpus. For ease's sake, we'll still assume that the player will use one of two strategies. Gold-first, or exit-first.

	Pot of gold	Rock	Wumpus
Entrance & Player			
		Water	
	Pot of gold		Exit

Before we talk about the 4x4 world, let's first take a look at the first example. If one were to explicitly define the conditional probability table, he/she needs to into account the possible world states, actions, and employed strategies. This would result in an entry count of #Actions * #Strategies * #WorldStates. The number of actions consist of moving in one of 4 directions, so #Actions = 4, whereas #Strategies would be 2, as we have the gold-first and exit-first strategies. Next, for the amount of world states, there 2x3 = 6 different locations. If one would take into account the player picking up the pot of gold resulting in different world states, this would add an extra 6 possible world states on top of that, making 12. So for all the combinations of P(action = X | strategy = Y, worldState = Z) one would require 4 * 2 * 12 = 96 different entries over which a distribution needs to be defined and computations need to be performed. When we add to this more states to visit, more gold to be collected that, when picked up, allow for more different world states, and the Wumpus itself, resulting in even more different world states as it can be located in any state as well, intractability is bound to follow.

So instead of this, we'll be taking a different approach to computing the conditional probabilities in the larger world. In order to make this work a more local method will be used, that is supposed to compute the probabilities based on the current player location. To do this, if-then rules are employed in order to implicitly, rather than explicitly, represent the probability distribution over the actions. We'll 'zoom in' on the player and check what the possible actions are, given the player's location. Following up, a number of functions will be executed that determine what strategies each actions complements. This is achieved by checking whether the action in question will put more, or less distance between the player and the nearest goal for the strategy in question. So if the sole pot of gold in a level is two locations east of the player, 'move east' will decrease the distance between, and as of such be an applicable action for gold-first. But if the exit happens to be east of the player as well, then the 'move east' action would be applicable to both strategies.

Next, each of these actions that complement a specific strategy will be assigned a value. This value is computed as followed: 1 / (n), where n is the number of actions that complement the strategy in question. Which is then computed for every 'action-strategy' combination. After that, every such value will be multiplied by the believed probability of the strategy. All in line with how the predicted probabilities were computed previously. Together, this results in the following bit of Python code when executed:

```
for strategy in strategies
        complementingActionCount = 0
        complementingActions = []
        for action in possibleActions
                if complementsStrategy(action, strategy)
                        complementingActions.append(action)
                        complementingActionCount += 1
        for action in complementingActions
                action.actionValue += (1/complementingActionCount) * stratProb
```

After execution of the code, all actions will have their predicted probabilities computed for them. This ensures that the predicted probability for each action that doesn't move the player closer towards the goal associated with the strategy to be 0, while the actions that do will be evenly distributed among each other. Keep in mind this is for that specific strategy only, as the action could also contribute to the other strategies, thus having its value increased.

5. Methods

5.1 Information trace of the AI

In order to provide extra clarity on the information the AI has, and how this is used, a number of screenshots will be shown. Alongside these screenshots traces of information corresponding with each screenshot are provided. These traces should clarify what information the Wumpus has, the paths it plans to take, and what it predicts the player will do. Not all the steps of a game are shown, as that would clutter multiple pages. The probability distributions are assigned as follows: for the strategies it is [p(gold-first), p(exit-first)], whereas for the actions it is [p(north), p(east), p(south), p(west)].

```
| None |     | GoldPot |     | Rock |     | Wumpus |
| Entrance  Player |     | None |     | None |     | None |
| None |     | None |     | Water |     | None |
| None |     | GoldPot |     | None |     | Exit |
```

A: Turn 1

At the first turn, there is no LOS yet between the Wumpus and player, as this is obstructed by the rock left of the Wumpus. Therefore there won't be any calculations on what the player is going to do. First and foremost because the Wumpus doesn't yet realize the player has entered its lair. Second because the Wumpus does not yet have information of any kind on the player, and therefore will be unable to perform the calculations. As of such, probability distribution will be the initial [0.5, 0.5] for the strategies, and [0.25, 0.25, 0.25, 0.25] for the actions. For its turn, the Wumpus will make a random move.

```
| None |     | GoldPot |     | Rock |     | None |
| Entrance |     | None |     | None |     | Wumpus |
| Player |     | None |     | Water |     | None |
| None |     | GoldPot |     | None |     | Exit |
```

B: Turn 2

With player and Wumpus both having moved south, the field looks like depicted above. As there are no vision-blocking objects in the path from Wumpus to Player (the path [(3, 1), (2, 1), (1, 1)], Wumpus and player have made visual contact with each other. Because the pot of Gold at location (1, 0) initially was closest to the player, the AI guessed this location to be its main target. Although after observing the player to have moved south instead, and thus moving away from the nearest pot of gold, the exit-first strategy seemed more likely to it. As for the next action it predicts the player to take, 'move east' is the most likely within the distribution of [0.01, 0.5, 0.49, 0]. Even though this is supposed to be on par with 'move south', as both actions would bring it closer towards the pot of gold, as well as the exit, so this was likely caused due to some miscalculations. Lastly, the path the AI now intends to take is [(3, 1), (2, 1), (1, 1), (1, 0)], this seems to still be based on the assumption that the player will go for the pot of gold at location (1, 0).

```
| None |     | GoldPot |     | Rock |     | None |
| Entrance |     | None |     | Wumpus |     | None |
| None |     | Player |     | Water |     | None |
| None |     | GoldPot |     | None |     | Exit |
```

C: Turn 3

Doing exactly as the AI expected, the player has indeed moved east, confirming its predictions. As there still are no vision-blocking objects between player and Wumpus, they can still see each other clear as day. Interestingly, although likely due to some erroneous coding, the Wumpus still assumes location (1, 0) to be the player's goal, even though the nearest pot of gold would now be at location (1, 3). As of such, the predicted action is now for the player to move north, even though this is not the most likely move that the player will make, judged by the new probability distribution over the actions, which is [0.24, 0, 0.72, 0.03] (which really should be [0.24, 0.02, 0.72, 0.02]). Although its assumptions about the player's goals are a bit off, it did now correctly identify the possibility of the player employing a different strategy, as the probability distribution over the strategies has become [0.5, 0.5], thereby acknowledging the player may go for the exit instead.

```
Information the AI has:
AI has LOS to the player:  True
Line of sight path to new location to see whether AI has LOS:  [(2, 1), (1, 2)]
Line of sight path to old location to see whether AI has LOS:  [(3, 1), (2, 2)]
Goal the AI guesses the player to have:  [3, 3]
Action the AI predicts the player to take:  EAST
Probability distribution over the strategies:  [0, 1]
Probability distribution over the actions:  [0.02, 0.98, 0, 0.02]
Path the AI intends to take:  [[3, 1], [3, 2], [3, 3]]

| None |     | GoldPot |     | Rock |     | None |
| Entrance |     | None |     | None |     | Wumpus |
| None |     | None |     | Water |     | None |
| None |     | Player |     | None |     | Exit |
```

D: Turn 4

Defying the AI's expectations, the player chose to move south instead and pick up the pot of gold located there. The AI now realizes the player probably isn't going for the gold, but is heading straight for the exit, as the predicted goal of the player has become (3, 3); which is the location of the exit. Furthermore, the AI now also predicts the employed strategy to be exit-first, seen from the distribution of [0, 1]. In accordance with this are the predictions of the actions, which are [0.1, 0.98, 0, 0.1], assuming the player will make a rush for the exit. Based on these assumptions, the Wumpus makes an attempt to intercept the player by moving east, which is the faster way to the exit. Unfortunately for the Wumpus, the player will have reached the exit in two more turns, thereby escaping the dungeon with the collected gold, and leaving the Wumpus hungry for the oncoming winter.

5.2 Program mechanisms

In this part more explanation is provided on the game so that the reader has at least general of what has been done. Starting off with a table in which all the used classes and their general function are listed, further explanation will be provided on the most important functions and components of the game.

Class	Description and task	Important functions
ActionHandler	Handles the move and attack actions, and whether movement in a given direction is possible.	/
ActionProbDistr	Calculates the probability distribution over the actions based upon if-then rules.	makeAPD(self), predictMove(self)
AIBehavior	Performs a variety of functions that collect information on the player and uses these to create the AI behavior	takePredictiveTurn(self)
Bresenham	Used as the basis for the line of sight mechanism	/
Character	Superclass that the PlayerCharacter and Wumpus classes extend. Holds information about the player and Wumpus.	/
GameField	Holds information about the playing area of the game.	/
GameRunner	Starts up everything necessary to play the game and receives user input.	/
GameTurn	Ensures constant flow of the game until it is finished.	/
Main	The main file that calls sets everything in motion	/
Model	Head-class that holds all the important information within the program.	hasLOSTo(self)
Pathfinder	Responsible for creating paths for the AI to traverse	createLocationList(self, targetLoc)
PredictionPlanner	Performs all sorts of checks to aid in prediction and planning of movement.	/
StratProbDistr	Holds the values of the probability distribution over the strategies.	/

ActionProbDistr:

- makeAPD(self): This function is used to create an updated probability distribution over the actions. It does so by taking the two strategies, gold-first and exit-first, and counting how many of the possible actions a character can take bring that character closer to the goal associated with that strategy. Next is determined which exact actions actually complement each strategy.
The information from these two separate functions are then used in combination with the probability values of the strategies to calculate the new probability values of the actions. See appendix (A) for more detail.

- predictMove(self): Used to let the AI decide what action he predicts the player to take. It takes the probabilities of the actions and makes it so that when a random number from 1 to 100 is taken, the probability that a specific action is chosen as final predicted action corresponds to those in the probability distribution. See appendix (B) for the code.

AIBehavior:

- takePredictiveTurn(self): Lets the AI take a term using predictive processing. Depending on a number of conditions, the AI will make a different action.
    * When the player hasn't been spotted at least once, it will make a random move to simulate natural behaviour.
    * If it is adjacent to the player, it will attack the player.
    * If it has line of sight to the player and hasn't visited the predicted player goal at least once, it will predict the player's goal location, like the exit, and move towards it to intercept the player. This predicted goal can change as the player makes an action.
    * When the AI has visited the predicted player goal at least once and has line of sight to the player, it will predict the location the player will go to and move towards that place.
    * Should these things all fail, meaning the Wumpus has lost sight of the player but has seen it disappear, it will move to the last location it saw the player. Should the player still not be visible after that, it will continue random movement until the player has been found.

Model:

- hasLOSTo(self): Used to determine whether the Wumpus can see the player and whether the player has been spotted at least once during a game. To do this, it uses the Bresenham's line algorithm to draw a line between the previous player location and the Wumpus' location, and between the new player location and the Wumpus' location.

Pathfinder:

- createLocationList(self, targetLoc): A* search is employed to find the optimal path towards the given target location. Successors are generated from the legitimate moves the Wumpus can take, to prevent the Wumpus from running through rocks. When a successor is found that leads to targetLoc, that successor is returned. This successor next gets taken apart by taking its parents one by one and placing their locations in a list to create the actual path coordinates.

5.3 Anecdotal pilot setup

As this project was more explorative rather than empiric, I have performed an anecdotal pilot instead of an actual experiment. The main reason for this was to see whether it would make sense to set up an actual formal experiment. Because of this, the following section should not be seen as a formal experiment, but as an explorative pilot that may provide clues as to whether it would be interesting for actual research to be performed on this topic. In this part, participants were asked to play the game, observations were made during play, and afterward feedback was received. Based on which some assumptions could be made.

Five persons were asked to play the game and provide some remarks. All of the participants were around 21 years old, and had at least some experience with playing games. Participants have gotten general information about the game, including the presence of two types of AI. Afterward, they had gotten more explanation about the AI's, and what the predictive AI was supposed to do.

Players first went through an introductory round against one of two AI's; the simple AI that chases the player down, or the predictive processing AI (randomly determined), to warm up and get used to the game. This was followed up by two sets of two games. In each set a different AI was used, the order of which was also randomly determined. What exact type of AI was being played against was hidden from view, in order to get non-biased answers to some of the questions.

Once the game has been played, and the extra information provided, the participant was asked to provide some basic commentary. Furthermore, they were questioned about topics like whether the predictive AI seemed to try and predict player action, and whether that was judged to be intelligent behaviour.

6. Observations

This section contains a general rundown of how the participants played and perceived the game, as well as the provided commentary and feedback, and some anecdotal results that stood out from the rest. These are not to be taken as results from an experiment, but as informal observations.

6.1 Impressions of the gameplay

In general the gameplay was understood fairly fast by all participants, with the exception of the occasional confusion caused by the interface and lack of a more visual representation of the game world. This was mainly because of the spacing between every location; two objects were at the same location at the same time caused the other locations in the same row of the playing field to be shoved to the right. Resulting in confusion for the players as to where the Wumpus was located. Furthermore, apart from the occasional crash when the predictive AI was calculating, the game played rather fluidly and the participants seemed to have at least some fun.

Even though players were told that collection of all pots of gold was not a necessity to complete a game, all players made attempts to gather all the gold anyway, with three players repeatedly taking risks to get all of the gold. Five out of five players made a distinction between two different types of AI, whereas four out of five correctly identified type A to be the simple or aggressive AI, and type B to be the predictive AI. All of the players experienced minor issues playing against the simple AI in comparison with the predictive AI.

6.2 Feedback and commentary

All players found the game to be fun and interesting to play, with the exception of the player who used the optimal path and seemed to have experienced less fun but found it interesting nonetheless. Four out of five player found the interface to be confusing and unclear, of which 2 commented that they would have preferred a more accurate visual representation with images. Players found the simple AI easy to play against, whereas they experienced moments of struggle against the predictive AI.

Players found both types of AI to be fun to play against, with the simple AI providing a sense of superiority as players led it astray and often successfully escaped the dungeon with the gold, and the predictive AI providing a challenge to the players. Players unanimously found the predictive AI to display more intelligent behaviour than the simple AI. With the exception of the player who took the optimal route (who couldn't very well differentiate between the two), all players decidedly found the predictive AI to be more fun and interesting to play against , making comments like 'Damn, you're smart', and 'That guy knows all of my tricks!'. Apparently, the predictive AI was not liked solely because it was harder, but also because of the mind games involved in competing against it, the feeling that the predictive AI had actual intelligence, and the need to adapt one's strategy. One player commented that the prediction of the predictive AI was sometimes too good, as even when the player broke the employed patterns, the AI still made predictions that struck the player like it knew where the player was going.

On the other end, two out of five players expressed moments of annoyance at the predictive AI when it correctly predicted where the player was going and intercepted the player, or when it preventing the player from going past it. In one case the annoyances seemed to be a positive kind, as it posed an interesting obstacle to overcome, whereas the other player didn't always approve of being denied passage.

6.3 Anecdotal results

Interestingly, three of the players, once having realized the general mechanisms of the simple AI, repeatedly moved north to south, or east to west when the simple AI was blocking off the passage. They waited until the AI made the mistake of entering the square the player previously stood instead of moving together with the player in the same direction, enabling the player the escape. Although the direction the Wumpus would go to when diagonally adjacent to the player was randomized (as long as the movement would bring it closer to the player), this caught my eye. The other two players tackled the AI in a different way, with one of the attempting to avoid the Wumpus altogether. The other player quickly managed to find a highly efficient path that allowed collection of both pots of gold, while preventing the Wumpus from coming close. The player found out that the rock behind which the Wumpus starts blocked vision, and thus first moving up to snatch the pot of gold, and subsequently going down to snatch the other allowed a good amount of moves to be executed without the Wumpus spotting the player. This resulted in a rather easy experience, and possibly contributed to the fact that this same player incorrectly identified the type A to be the predictive AI. This was likely caused due to the decreased amount of behaviour the player saw from the AI, as the player kept abusing this path.

During one round, one of the players, after having reached the southern border of the field, kept attempting to go south until the player in question got eaten by the Wumpus. The player subsequently was confused about what happened; apparently the player had assumed that the game field as it was presented was not the entire world, and that the field would 'scroll down' to reveal more of the world.

Two players expressed surprise, and/or possibly confusion, when the AI didn't immediately go for the player but instead headed for the exit. They seemed to be more interested into analysing the AI's movements as this kind of behaviour was new to them.

On some occasions the players got themselves cornered by the AI, with the AI preventing the player from moving past it. With the simple AI, as described above, some players repeated movements until it made a mistake. In the case of the predictive AI, though, this strategy proved fruitless as it still prevented the player from advancing, resulting in some players giving up trying to get past and instead attacking the Wumpus.

7. Conclusion

Even though this was no empiric research, some conclusions can be drawn from the information that has been gathered. These aren't factual statements, but statements based on experiences and comments from a small group of participants. While unable to serve a proof for theories, they may pique the interest of other researchers and open up ways to more research in this area.

- As the majority (four out of five) participants liked the predictive AI better than the simple AI, it can be said that a predictive game AI would be a well-received feature of a game from a player perspective. Although this would still depend on the type of game, as not every game would benefit from this.

- All players unanimously found the predictive AI to display more intelligent behaviour, therefore it can be concluded that predictive processing can be used to create a more intelligent AI in gaming.

- All of the players found the predictive AI to behave more realistically and human-like than the simple AI. From this it can be stated that predictive processing indeed can be used to produce a more realistic AI in games.

Based on these three observations, it can at least be said that predictive processing in games may very well be an interesting thing, as it was very positively received by the players. As of such, this topic deserves further research in order to see whether this can indeed become a very beneficial thing to science and the gaming industry.

8. Discussion

All in all, it seems that using predictive processing can be a great addition in game AI, at least for a dungeon crawler game. Although is one point that possibly undermines the legitimately of the results, apart from the low number of participants. Namely the fact that all of the participants were either friends or relatives of mine. This could have led to them unconsciously rating the predictive AI better than the simple AI as they might have thought that rating that AI higher would be more beneficial to me.

Furthermore the player who took the highly efficient route stood out in that this participant mistakenly thought the type A AI to be the predictive version, and type B to be the simple version. It is likely that this was likely caused due to the reduced action this player had seen from both types. Interestingly, this player had also experienced less fun than the others, quite possibly another effect from having taken a 'quick and dirty' route.

9. Further Research

- The study could be redone in a more empirical fashion, with a larger group of participants. This would allow for more factual conclusions to be made.

- On the topic of human behaviour, one could look into the different human responses when facing adversity. Players could face the simple AI first, and when faced with the harder predictive AI, be monitored on their responses.

- An interesting question for the gaming industry would be whether players prefer simple opponents, hard opponents created by cheating AI, or hard opponents created by intelligent AI. A lot of games nowadays are more streamlined and easier than their predecessors in order to make them more accessible. But maybe it would be beneficial to have the option of employing intelligent AI.

- For cognitive neuroscience, brain activity in players playing the game can be measured in order to see what brain areas are active during the combination of gaming and strategy planning.

10. References

- AI game bots ‘more human-like’ than half of human competitors September 27, 2012, http://www.kurzweilai.net/ai-game-bots-more-human-like-than-half-of-human-competitors

- Brown, H., Friston, K. J., & Bestmann, S. (2011). Active inference, attention, and motor preparation. Frontiers in psychology, 2, 218.

- Clark, A. (2012). Dreaming the whole cat: Generative models, predictive processing, and the enactivist conception of perceptual experience. Mind, fzs106.

- Friston, K. (2002). Functional integration and inference in the brain. Progress in neurobiology, 68(2), 113-143.

- Friston, K. (2003). Learning and inference in the brain. Neural Networks, 16(9), 1325-1352.

- Friston, K., Adams, R., Perrinet, L., & Breakspear, M. (2012). Perceptions as hypotheses: saccades as experiments. Frontiers in psychology, 3, 151.

- Kwisthout, J., Bekkering, H., & van Rooij, I. (2016). To be precise, the details don’t matter: On predictive processing, precision, and level of detail of predictions. Brain and cognition.

- Milchtaich, I. (1996). Congestion games with player-specific payoff functions.*Games and economic behavior*, *13*(1), 111-124.

- Nareyek, A. (2004). AI in computer games. *Queue*, *1*(10), 58.

- Wang, D., Subagdja, B., Tan, A. H., & Ng, G. W. (2009, July). Creating human-like autonomous players in real-time first person shooter computer games. In*Proceedings, Twenty-First Annual Conference on Innovative Applications of Artificial Intelligence* (pp. 173-178).

- Yannakakis, G. N. (2012, May). Game AI revisited. In *Proceedings of the 9th conference on Computing Frontiers* (pp. 285-292). ACM.

- Yoshida, M., Dickey, M. W., & Sturt, P. (2013). Predictive processing of syntactic structure: Sluicing and ellipsis in real-time sentence processing. Language and Cognitive Processes, 28(3), 272-302.

11. Appendix

11.1 A

```
# Computes the new action probability distribution
def computeActionVals(self):
    self.actionVals = [0, 0, 0, 0]
    self.stratProbDistr.updateStrategies()
    self.sPD = self.stratProbDistr.strategyVals
    CONST_DIRECTIONS = ['NORTH', 'EAST', 'SOUTH', 'WEST']
    for dir in CONST_DIRECTIONS:
        if dir == 'NORTH':
            if dir in self.actionsComplementingGF:
                self.actionVals[0] +=
abs((float(1)/float(len(self.actionsComplementingGF))) * float(self.sPD[0])
- (0.01 * (len(self.movesPossible) - len(self.actionsComplementingGF))))
            if dir in self.actionsComplementingEF:
                self.actionVals[0] +=
abs((float(1)/float(len(self.actionsComplementingEF))) * float(self.sPD[1])
- (0.01 * (len(self.movesPossible) - len(self.actionsComplementingEF))))
            if (dir not in self.actionsComplementingGF) and (dir not in
self.actionsComplementingEF) and (dir in self.movesPossible):
                self.actionVals[0] += abs((0.01 *
len(self.actionsComplementingEF)) + (0.01 *
len(self.actionsComplementingGF)))
        elif dir == 'EAST':
            if dir in self.actionsComplementingGF:
                self.actionVals[1] +=
abs((float(1)/float(len(self.actionsComplementingGF))) * float(self.sPD[0])
- (0.01 * (len(self.movesPossible) - len(self.actionsComplementingGF))))
            if dir in self.actionsComplementingEF:
                self.actionVals[1] +=
abs((float(1)/float(len(self.actionsComplementingEF))) * float(self.sPD[1])
- (0.01 * (len(self.movesPossible) - len(self.actionsComplementingEF))))
            if (dir not in self.actionsComplementingGF) and (dir not in
self.actionsComplementingEF) and (dir in self.movesPossible):
                self.actionVals[1] += abs((0.01 *
len(self.actionsComplementingEF)) + (0.01 *
len(self.actionsComplementingGF)))
        elif dir == 'SOUTH':
            if dir in self.actionsComplementingGF:
                self.actionVals[2] +=
abs((float(1)/float(len(self.actionsComplementingGF))) * float(self.sPD[0])
- (0.01 * (len(self.movesPossible) - len(self.actionsComplementingGF))))
            if dir in self.actionsComplementingEF:
                self.actionVals[2] +=
abs((float(1)/float(len(self.actionsComplementingEF))) * float(self.sPD[1])
- (0.01 * (len(self.movesPossible) - len(self.actionsComplementingEF))))
            if (dir not in self.actionsComplementingGF) and (dir not in
self.actionsComplementingEF) and (dir in self.movesPossible):
                self.actionVals[2] += abs((0.01 *
len(self.actionsComplementingEF)) + (0.01 *
len(self.actionsComplementingGF)))
        elif dir == 'WEST':
            if dir in self.actionsComplementingGF:
                self.actionVals[3] +=
abs((float(1)/float(len(self.actionsComplementingGF))) * float(self.sPD[0])
- (0.01 * (len(self.movesPossible) - len(self.actionsComplementingGF))))
```

```
            if dir in self.actionsComplementingEF:
                self.actionVals[3] +=
abs((float(1)/float(len(self.actionsComplementingEF))) * float(self.sPD[1])
- (0.01 * (len(self.movesPossible) - len(self.actionsComplementingEF))))
            if (dir not in self.actionsComplementingGF) and (dir not in
self.actionsComplementingEF) and (dir in self.movesPossible):
                self.actionVals[3] += abs((0.01 *
len(self.actionsComplementingEF)) + (0.01 *
len(self.actionsComplementingGF)))
```

11.2 B

```
# Predicts what move the player will make for its next action
def predictMove(self):
    self.makeAPD()
    randomNum = random.randint(1, 100)
    prob1 = self.actionVals[0] * 100
    prob2 = prob1 + (self.actionVals[1] * 100)
    prob3 = prob2 + (self.actionVals[2] * 100)
    prob4 = prob3 + (self.actionVals[3] * 100)
    if randomNum <= prob1:
        return 'NORTH'
    elif randomNum <= prob2:
        return 'EAST'
    elif randomNum <= prob3:
        return 'SOUTH'
    elif randomNum <= prob4:
        return 'WEST'
```