Machine Generated Intelligent Agents for Games with Simultaneous Movement

Jeremy Keffer

March 26, 2013

1 Introduction

From the early days of AI, computers have been programmed to play games against human players [22, 23]. That continues to this day [38, 32, 19, 7, 33, 25, 35, 26, 4, 39]. Most of the AI work has sought to build world-champion programs to play turn-based games such as Chess and Checkers [5, 32]. But since it's beginnings in the late 1970s, computer gaming has become more and more prevalent as a form of recreation/entertainment. In 2009, it was reported that computer and video games was a ten and a half billion dollar industry [10]. In contrast to the turn-based two-player games that AI originally studied, the two-player and multi-player games developed by the computer gaming industry increasingly provide for *simultaneous movement*.

Simultaneous movement was seldom used in games before the advent of computers because of the bookkeeping complexities it introduces. But now that everyone has a computer, simultaneous movement games are more common. They are more common because in a sense they are more realistic; in war and business competition, the participants don't take turns, they act as soon as they know what they want to do. There is work being done in the Artificial Intelligence community that involve this more realistic action, but a) it is still in its early phases and b) it appears to have ignored what classical game theory has to say about simultaneous movement games [9, 6].

In this proposal we present an adaptation of *Recursive Game Theory* which can be used to analyze computer games. Recursive game theory deals with simultaneous movement games with many (possibly looping) states, and we believe this often overlooked branch of game theory to be incredibly applicable to modern computer games. In Section 2 we describe the premise of several example games which have properties found in modern games, as well as give a complete formalization of these games. In Section 3 we briefly discuss the relevant aspects of game theory in general and Recursive game theory specifically required to understand our application of the theory. We then describe our adaptation of recursive game theory in 4. Finally, we propose two possible systems for generating game AI programs which apply our adaptation of the theory: 1) an offline planner which produces a program which follows a *universal plan* for the input game in Section 5, and 2) an online planner which uses recursive game theory to make decisions on the fly during game play in Section 6.

2 Game Collection

In this section we provide descriptions of several games used in this proposal. The premise of each game is given, followed by a formalization of the game.

2.1 Right Turn

Right Turn is a minor variation on the *Hamstrung Squad Car* game first proposed in [18]. This game has the property that the orc can't win when the game starts from certain states and the monkey plays optimally. Nevertheless, when the game is at one of these states, some orc actions are preferable to other actions. Consideration of these preferences leads to our *Game Theory of Lowered Expectations* in Section 4.2. In Section 2.1.1 we describe the premise of the game. We then describe formally the rules of the game in Section 2.1.2.

2.1.1 Premise

Thok Fump is a soldier in the High Orc Guard. Unfortunately today his pet capuchin monkey Lil' Cal decided to run away whilst the two were outside playing. To further complicate matters, today is right turn Friday! That is, it's against the law in Thok's village to turn left (and it's not a good idea to break orc laws). On the bright side, Thok's training has rendered him an excellent sprinter (so he's able to outrun Lil' Cal). As an orc, Thok isn't very smart - so he needs our help to catch his Lil' Cal.

2.1.2 Rules

The naive way to represent the state of this game would require five parameters: two parameters to locate the position of the monkey, two parameters to locate the position of the orc, and one parameter to show the orc's orientation (facing North, South, East, or West). The number of parameters can be reduced to two by considering any states that can be transformed into each other by translations and/or $90°$ rotations as the same state. That state is represented by the single configuration where the monkey is at the origin of the Cartesian plane and the orientation of the orc is to be facing North. The only remaining parameters that are needed to specify a state are then the coordinates of the orc relative to the monkey. (This representation is similar to the one used by Rufus Isaacs[18].)

A State in Right Turn is a vector (x, y) such that $x, y \in \mathbb{Z}$; and (x, y) represents the position of the orc facing North on the cartesian plane relative to the position of the monkey.

The following classes of states are terminal and are win states for the orc:

$$(x, y) \ni |x| \leq 1 \wedge |y| \leq 1 \tag{2.1}$$

There are no win states for the monkey. The monkey's only goal is to keep the game going.

Informally, the game ends when the monkey is within arm's reach of the orc - that is, when the orc is in the monkey's *Moore neighborhood* [17].

Possible actions at each state for the orc are to either move forward two steps or to turn right and move forward two steps. These shall henceforth be referred to as F and R, respectively. Possible actions for the monkey in any state are to move North (N), South (S), East (E), or West (W) one step.

Let α be the action chosen by the orc and β the action chosen by the monkey. Then, for state $\gamma = (x, y)$ the next state γ' is obtained via Algorithm 1.

In English: when the monkey moves, the result is that the orc is displaced by one unit in the opposing direction. The orc, who is always left facing North, will move North by two units when choosing to move foward. When the orc chooses to turn right, he will move two units to the East and the entire board is rotated counterclockwise 90 degrees around the monkey (orienting the orc Northward again).

Algorithm 1 Transition function for Right Turn

```
$(x', y') \leftarrow (x, y)$

if $\beta = N$ then
    $y' \leftarrow y' - 1$
else if $\beta = S$ then
    $y' \leftarrow y' + 1$
else if $\beta = E$ then
    $x' \leftarrow x' - 1$
else if $\beta = W$ then
    $x' \leftarrow x' + 1$
end if

if $\alpha = F$ then
    $y' \leftarrow y' + 2$
else if $\alpha = R$ then
    $t \leftarrow x' + 2$
    $x' \leftarrow -y'$
    $y' \leftarrow t$
end if

return $(x', y')$
```

2.2 Bear Brawl

Bear Brawl is a class of simple, perfect information games which embody many of the basic concepts found in fantasy games played by humans. The premise of the game is explained in Section 2.2.1. In Section 2.2.2 we describe a simple instance of Bear Brawl which we will use to illustrate our theory in the later sections.

2.2.1 Premise

In an exhortion attempt The Evil Wizard Bertram cast a devious spell upon the kingdom of Quahogeroth. Unless King Stewie hands over the entire contents of the royal treasury to Bertram, all of Quahogeroth will remain under Bertram's spell whereupon they will dance to Aqua's 1997 single "Barbie Girl" in perpetuity.

Distraught, King Stewie sought the help of a band of noble warriors to seek out the magical antidote to Bertram's spell. As it turns out, the main ingredient to the antidote is a magical powder which can be harvested from Phoolabairs Forest. Unfortunately, however, the bears indigenous to the forest will guard the powder with their lives as they enjoy sniffing it to get high.[1]

Can our heroes brave a forest of cracked out bears in order to save Quahogeroth from an eternity of getting down to cheesy 90s techno? Will the bears be able to sucessfully

[1] Any resemblence between the magic powder and any real mind-altering substance used by humans is purely coincidental.

guard their precious mind-altering substance from outside looters? Through a combination of game theory, machine learning, and heuristic search we hope to answer these questions and more.

2.2.2 1D Hero vs. Bear

In this section we introduce the rules of a variation of Bear Brawl we refer to as the *One-Dimensional one hero vs. one bear problem* (1D1H1B).

This variation is played on a (possibly infinite) one-dimensional board indexed by $\mathbb{Z}$. The powder is always located at location 0. The game progresses as a series of *rounds* such that on each round both the hero and the bear (independently) choose one of *Move left*, *Move right*, or *Attack*.

A state in 1D1H1B is a vector (u, v, x, y) such that $u, v \in \mathbb{N}$, $x, y \in \mathbb{Z}$; and u represents the *hit points* of the hero, v the hit points of the bear, x the location of the hero, and y the location of the bear.

The following classes of states are terminal and are win states for the hero:

$$(u, v, 0, y) \ni u, v > 0 \wedge y \neq 0 \tag{2.2}$$

$$(u, 0, x, y) \ni u > 0 \tag{2.3}$$

The following class of states are terminal and are win states for the bear:

$$(0, v, x, y) \tag{2.4}$$

Informally, The hero wins either when she[2] obtains the powder (her position is 0 and the bear's is not), or when she kills the bear (the bear's HP reaches 0). If at any point the hero dies (her HP reaches 0), the bear is declared the victor.[3]

Possible actions at any state for either player are to move left, move right, or attack. These shall henceforth be referred to as L, R, and A, respectively.

Let α be the action chosen by the hero and β the action chosen by the bear. Then, for state $\gamma = (u, v, x, y)$ the next state γ' is obtained via algorithm 2. In English, moving left or right will cause your location to decrement or increment, respectively. Attacking when your opponent occupies the same location as yourself will cause your opponent to lose one hit point. When resolving actions to determine the next state, moves resolve before attacks - *i.e.* a player can walk into an attack or jump out of the way of one.

3 Game Theory Background

In this section we will describe and discuss the background necessary in understanding our approach to games. We begin in subsection 3.1 by describing necessary foundational theory in two-player zero-sum games. We then introduce the concept of a *Recursive Game* in subsection 3.3, which is the starting point for our approach.

[2]Our hero in this game is a woman. The bear, however, is male. A big, angry, tweaked male bear.

[3]Note that the bear may go down a martyr if he and the hero die at the same time.

Algorithm 2 Transition function for 1D1H1B

```
$(u', v', x', y') \leftarrow (u, v, x, y)$

**if** $\alpha = L$ **then**
    $x' \leftarrow x' - 1$
**else if** $\alpha = R$ **then**
    $x' \leftarrow x' + 1$
**end if**

**if** $\beta = L$ **then**
    $y' \leftarrow y' - 1$
**else if** $\beta = R$ **then**
    $y' \leftarrow y' + 1$
**end if**

**if** $\alpha = A \wedge x' = y'$ **then**
    $v' \leftarrow v' - 1$
**end if**

**if** $\beta = A \wedge x' = y'$ **then**
    $u' \leftarrow u' - 1$
**end if**

**return** $(u', v', x', y')$
```

3.1 Two-Player, Zero-Sum Games

The theory in this section is derived from [30]. For a more in depth discussion of game theory and two-player, zero-sum Games see [30], [36] or [24].

A *two-player, zero-sum Game* Γ is an object with the structure:

$$\Gamma = (N, C, u) \tag{3.1}$$

where N is the set $\{P_1, P_2\}$ of *players*, C is a vector (C_1, C_2), where C_1 and C_2 are the sets of available *actions* for player 1 and player 2, respectively, and u is a function $C_1 \times C_2 \mapsto \mathbb{R}$ which is the *payoff function*.[4]

As an example, consider the MacWiggins brothers, Sven and Olaf. It's a typical evening at the bar for these two. The alcohol has been flowing freely, and they have again gotten into a heated argument over who has the finest, most full bodied beard. These arguments over facial hair generally culminate in fisticuffs, and tonight was no exception. Now Sven, who is ambidexterous (and thus is capable of punching equally hard with either arm,) has Olaf up against the wall. Olaf, not knowing from which direction his brother's blow will come, has the option of dodging either left or right.

If Sven punches with his right hand and Olaf dodges to the right, the punch will connect and Olaf will end up with a black eye. The outcome will be very similar if Sven punches with his left hand and Olaf dodges to the left. However, if Sven punches with his right hand and Olaf dodges to the left or vice-versa, Sven will miss his brother and hit the wall, only to end up with a broken hand.

[4]N.B. that one of the players (generally 1) is the *maximizing player* with the goal of maximizing u. The other player is the *minimizing player* with the goal of minimizing u.

We can model the MacWigginses' situation as the two-player, zero-sum game $(\{1,2\},(\{PL,PR\},\{DL,DR\}),u)$ where Sven is player 1, Olaf player 2, PL and PR stand for punch left and punch right, respectively, DL and DR stand for dodge left and dodge right, respectively, and u is defined by the matrix in (3.2).

$$\left[\begin{array}{c|cc} u & DL & DR \\ \hline PL & 1 & -1 \\ PR & -1 & 1 \end{array}\right] \tag{3.2}$$

Note that, as the maximizing player, Sven's most favorable outcomes are (PL,DL) and (PR,DR) - exactly the outcomes in which his fist makes contact with his brother's skull. Olaf as the minimizing player prefers (PL,DR) and (PR,DL) - which correspond to those outcomes in which Sven ends up with bloody knuckles.

3.2 Solving Two-Player, Zero-Sum Games

A *strategy* σ_i for player i is defined to be a probability disribution over C_i - that is, $\sigma_i : C_i \mapsto \{x \in \mathbb{R} | 0 \le x \le 1\}$ and $\sum_{c \in C_i} \sigma_i(c) = 1$. A strategy σ_i is said to be *pure* if $(\exists c \in C_i)[(\sigma_i(c) = 1) \wedge (\forall c' \in C_i)(c' = c \vee \sigma_i(c') = 0)]$, *i.e.* a *pure strategy* is one in which the player chooses a specific action with 100% certainty. A strategy σ_i which is not a pure strategy is a *mixed strategy*, in which the player i stochastically chooses his/her action using the probability distribution defined by σ_i. Let $\Delta(C_i)$ be the set of all possible strategies ranging over C_i.

Going back to our farmer siblings, $(1,0)$ would be the pure strategy for Sven in which he always chooses to punch with his left hand. Olaf could counter this strategy with the pure strategy $(0,1)$, in which case he would always dodge to the right. An example mixed strategy for Sven is $(0.5,0.5)$ - meaning he will choose to deliver a right hook with 50% probability, and a left hook the other 50%.

For a two-player, zero-sum game Γ, Let $\sigma = (\sigma_1, \sigma_2)$, where σ_1 and σ_2 are strategies for players 1 and 2, respectively, be a *mixed strategy profile*. The *utility* $u'(\sigma)$ of σ is defined to be:

$$\sum_{c_1 \in C_1} \sum_{c_2 \in C_2} \sigma_1(c_1) \cdot \sigma_2(c_2) \cdot u(c_1, c_2) \tag{3.3}$$

A mixed strategy profile σ is a *Nash Equilibrium* iff

$$(\forall \tau \in \Delta(C_1))[u'(\sigma) \ge u'((\tau, \sigma_2))] \wedge (\forall \tau \in \Delta(C_2))[u'(\sigma) \le u'((\sigma_1, \tau))] \tag{3.4}$$

That is, neither player could increase her/his expected payoff by unilaterally deviating from her/his strategy as specified by σ. [5]

To find a Nash Equilibirum σ for a game Γ, one must solve the following pair of dual optimization problems:

$$\begin{gathered} \text{maximize } v \text{ subject to:} \\ (\forall c_2 \in C_2)\left(v \le \sum_{c_1 \in C_1} u(c_1, c_2) \cdot x_{c_1}\right) \\ \sum_{c_1 \in C_1} x_{c_1} = 1 \\ (\forall c_1 \in C_1)(x_{c_1} \ge 0) \end{gathered} \tag{3.5}$$

[5]Note that equation (3.4) is the definition of a Nash Equilibrium when player 1 is the maximizing player and player 2 the minimizing. If the opposite case is true, the inequalities in (3.4) must be reversed.

and

$$\begin{aligned}
&\text{minimize } w \text{ subject to:} \\
&\qquad (\forall c_1 \in C_1)\left(w \geq \sum_{c_2 \in C_2} u(c_1, c_2) \cdot y_{c_2}\right) \\
&\qquad \sum_{c_2 \in C_2} y_{c_2} = 1 \\
&\qquad (\forall c_2 \in C_2)(y_{c_2} \geq 0)
\end{aligned} \tag{3.6}$$

The resulting vectors $(x_{c_1} | c_1 \in C_1)$ of (3.5) and $(y_{c_2} | c_2 \in C_2)$ of (3.6) give us σ_1 and σ_2, respectively, of σ. This result, including the fact that $v = w$ is the foundation of two-player zero-sum game theory. For more information on optimization, solving optimization problems using linear programming, and linear programming as it relates to two-player, zero-sum games see [40]. A very good discussion on two-player, zero-sum games as linear programming problems is also in [36].

The fundamental theorem about about two-person zero-sum games is that they all have Nash equilibria, and if there is more than one Nash equilibrium for a game, the value of the game is the same for each Nash equilibrium [3, 24, 30, 36]. For any game Γ, let $\mathcal{N}(\Gamma)$ be some Nash Equilibrium for Γ, and let $\mathcal{V}(\Gamma)$ be $u'(\mathcal{N}(\Gamma))$ or the *value* of Γ. In practice $\mathcal{N}$ and $\mathcal{V}$ are computed together. As it turns out, in the case of the MacWigginses, $((0.5, 0.5), (0.5, 0.5))$ is the only Nash Equilibrium profile. Both Sven and Olaf should choose randomly and evenly between their possible actions, and the value of the game is 0.

3.3 Recursive Games

Recursive games were originally postulated by Everett in [12].

A *recursive game* $\Re^\Gamma$ is defined as follows:

$$\Re^\Gamma = (\Gamma, \Gamma^T, N, C^\Omega, C, f, p) \tag{3.7}$$

Where N is as defined in equation (3.1); Γ is a set of *states*; $\Gamma^T \subset \Gamma$ is a set of *terminal states*; C^Ω is a tuple (C_1^Ω, C_2^Ω) where C_1^Ω and C_2^Ω are the sets of all available actions for player 1 and 2, respectively; C is a tuple (C_1, C_2) such that C_i for $i \in \{1, 2\}$ is a function $\Gamma \mapsto \mathcal{P}(C_i^\Omega)$ which gives the set of available actions for player i in a given state; f is a partial function $\Gamma \setminus \Gamma^T \times C_1^\Omega \times C_2^\Omega \mapsto \Gamma$ which is the *transition function* and satisifes the property $(\forall \gamma \in \Gamma \setminus \Gamma^T)(\forall c_1 \in C_1(\gamma))(\forall c_2 \in C_2(\gamma))(f(\gamma, c_1, c_2) \in \Gamma)$; and p is a function $\Gamma^T \mapsto \mathbb{R}$ which is the *payoff function*.

For a small example, let's consider the sister rogues Guan Damei and Guan Xiaomei.[6] Damei is an exceptionally talented archer, whereas Xiaomei prefers daggers and is a master of stealth. The sisters regularly hold practice sessions to keep up their position as the two best rouges in the land. In this particular session, Damei has only one arrow left to fire. If she fires it while her sister is hiding, she'll be a sitting duck for Xiaomei to knife in the back at her leisure. However, Xaiomei can pop out of stealth at any moment and surprise stab her sister. It's only when Damei chooses to fire her arrow at the same time her sister drops stealth that she'll come out on top (not even Xiaomei is fast enough to avoid Damei's perfect aim).[7]

[6]"Big Sister" and "Little Sister", respectively.

[7]To be clear, the sisters use prop weapons during training - the only harm done to the loser is to her pride.

We can model this particular session as the recursive game $(\{S, D, X\}, \{D, X\}, (\{W, R\}, \{W, F\}), (C_1, C_2), f, p)$ where S denotes the start of the session, D the situation where Xiaomei is pierced by the arrow, and X the situation where Damei gets knifed. Being designated as player 1, Xiaomei's available actions are to *wait* (W) or *run* (R). Likewise, Damei's available actions are to *wait* or to *fire* (F). C_1 is the function which gives the set $\{W, R\}$ on any input and C_2 the function which returns $\{W, F\}$ on any input. S being the only value possible as the first argument of f, the matrix in (3.8) defines f.

$$\left[\begin{array}{c|cc} S & W & F \\ \hline W & S & X \\ R & X & D \end{array}\right] \tag{3.8}$$

p is defined in (3.9).

$$p(x) = \begin{cases} 1 & \text{if } x = X \\ -1 & \text{if } x = D \end{cases} \tag{3.9}$$

3.4 Value of Recursive Games

In recursive games, the value of each game state is dependent on the values of the other game states. Given a game R^Γ as specified in (3.7) an assignment vector $\vec{v}$ of values to the game states in Γ, and a game state γ in $\Gamma \setminus \Gamma^T$, we can define two-player zero-sum game $\mathcal{G}(\gamma, \vec{v})$ to be

$$\mathcal{G}(\gamma, \vec{v}) = (N, (C_1(\gamma), C_2(\gamma)), \lambda x, y.\vec{v}[f(\gamma, x, y)]) \tag{3.10}$$

The matrix defining the utility function for this game is the matrix defining transitions from γ (represented by the function f with γ as the first argument) with each element (a game state in Γ) replaced by the value assigned to the element by $\vec{v}$. This game has a value as discussed in Section 3.2. Since $\mathcal{G}$ is defined on the non-terminal states and the payoff function p is defined on the terminal states, we can compute what the game value of each state must be assuming that the games states have been assigned the values specified by $\vec{v}$. Let $F(\vec{v})$ be the result of that computation. For γ in Γ,

$$F(\vec{v})[\gamma] = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ \mathcal{V}(\mathcal{G}(\gamma, \vec{v})) & \text{otherwise} \end{cases} \tag{3.11}$$

The only way that a value assignment $\vec{v}$ makes sense as a vector of game values for the game states in Γ is for it to be a fixed point of F, that is, the equation

$$\vec{v} = F(\vec{v}) \tag{3.12}$$

must be satisfied. It turns out that this equation always has a solution, but unfortunately, it can have many solutions.

In [12], Everett proved that every recursive game $\Re^\Gamma$ has a unique associated vector $\vec{v}$ indexed by Γ that not only is a fixed point of (3.12), but satisfies other desirable properties. He called $\vec{v}$ the *critical vector* for $\Re^\Gamma$. He further proved that knowing $\vec{v}$ is not enough in general to find a strategy such that the expected payoff matches the values in $\vec{v}$. However, he did prove that for any $\epsilon \in \mathbb{R}$ there is a strategy profile σ^ϵ such that for all $\gamma \in \Gamma$, the expected value of the game at γ is $\vec{v}[\gamma] \pm \epsilon$ for any player who plays their respective strategy in σ^ϵ. In other words, players can find strategies which will get them arbitrarily close to the value of the game, but cannot achieve an expected payoff equal to the value of the game in general. Such strategies are called ϵ-*strategies*.

In the case of the Guan sisters, the higher the probability with which both Damei and Xiaomei choose to wait, the closer the expected value of the game gets to 1 (a win for

Xiaomei). However, in the case where both rogues choose to wait 100% of the time, the game never ends and therefor neither woman wins (so the value of the game is effectively 0).

Thuijsman and Vrieze go a step further in [37] by constructively proving the existence of such ϵ-strategies. Iteratively applying

$$F'(\vec{v})[\gamma] = \begin{cases} F(\vec{v})[\gamma] & \text{if } \gamma \in \Gamma^T \\ \beta \cdot F(\vec{v})[\gamma] & \text{otherwise} \end{cases} \tag{3.13}$$

for some $\beta \in [0, 1)$ will eventually converge on some $\vec{v_\beta}$. Furthermore, $\lim_{\beta \to 1} \vec{v_\beta} = \vec{v}$. They proved that for all ϵ there exists a β such that $\mathcal{N}((N, (C_1(\gamma), C_2(\gamma)), u_{\gamma\beta}))$ is an ϵ-strategy profile for $\Re^\Gamma$ at $\gamma \in \Gamma \setminus \Gamma^T$ where

$$u_{\gamma\beta}(x, y) = \begin{cases} \vec{v}[f(\gamma, x, y)] & \text{if } \vec{v}[f(\gamma, x, y)] < 0 \\ \vec{v_\beta}[f(\gamma, x, y)] & \text{otherwise} \end{cases} \tag{3.14}$$

3.5 1D Hero vs. Bear as a Recursive Game

As a recursive game, 1D1H1B is

$$(\Gamma, \Gamma^T, \{h, b\}, (\{L, R, A\}, \{L, R, A\}), (C, C), f, p) \tag{3.15}$$

where Γ is the set of all tuples of the form $(u, v, x, y) \ni u, v \in \mathbb{N} \wedge x, y \in \mathbb{Z}$, Γ^T is the set of all states in Γ matching the descriptions in (2.2), (2.3), and (2.4), C is a function which returns $\{L, R, A\}$ on any input, f is the function defined by algorithm 2, and the payoff function p is defined as such:

$$p(\gamma) = \begin{cases} 1 & \text{if } \gamma \text{ is a win for the hero} \\ -1 & \text{otherwise} \end{cases} \tag{3.16}$$

We have determined that the game value vector $\vec{v}$ for this game is

$$\vec{v}[(u, v, x, y)] = \begin{cases} 1 & \text{if } |x| < |y| \vee (|x| = |y| \wedge u > v) \vee (|x| > |y| \wedge u > v + 1) \\ -1 & \text{otherwise} \end{cases} \tag{3.17}$$

Possible policies for this game (in geneal, games can have more than one optimal strategy profile for each game state) are shown in Algorithms 3 and 4.

Algorithm 3 Possible hero policy in 1D1H1B

```
function HEROACT($(u, v, x, y)$)
    if $0 = |x| = |y| \wedge u > v$ then
        return $A$
    else if $0 < |x| < |y|$ then
        return $R$ if $x < 0$ else $L$
    else if $0 < |x| = |y| \wedge u > v$ then
        return $R$ if $x < 0$ else $L$
    else if $|y| < |x| \wedge u > v + 1$ then
        return $R$ if $x < 0$ else $L$
    else
        return $R$ if $|x + 1 - y| > |x - y|$ else $L$
    end if
end function
```

Algorithm 4 Possible bear policy in 1D1H1B

```
function BEARACT($(u, v, x, y)$)
    if $0 = |y| = |x| \wedge v \geq u$ then
        return $A$
    else if $0 = |y| < |x| \wedge v \geq u - 1$ then
        return $A$
    else if $0 < |y| = |x| \wedge v \geq u$ then
        return $R$ if $y < 0$ else $L$
    else if $0 < |y| < |x| \wedge v \geq u - 1$ then
        return $R$ if $y < 0$ else $L$
    else
        return $R$ if $|y + 1 - x| > |y - x|$ else $L$
    end if
end function
```

4 Adaptation of Recursive Game Theory

So far we've been talking about how to extract a policy for playing to win. In fact, game theory is entirely premised upon the idea that all players are playing to win. However, our end goal is not necessarily to create an AI opponent who plays to win, but rather one who plays to make the game entertaining for the human player. One way to do this is for the computer to keep the game going by simply thwarting the human's attempt at winning. [36, 39]

This goal suggests a type of game analysis that is inspired by Rufus Isaacs' Differential Game Theory [18], specifically his analysis of what he called *Discrete Differential Games*. Discrete Differential Games are very similar to recursive games in that they consist of multiple states in which some are terminal. However, these games are limited in that only one of the players has a choice of action at any given state. Isaacs' method for solving these games involves assigning a payoff value of 0 to states in which player 1 wins the game. He then works backward from terminal states, assigning a value of 1 to states which are exactly 1 move away from some terminal, 2 to those which are 2 moves away, and so on and so forth. The end result is that the value of the game at any given state γ is the time (in combined actions for players 1 and 2) which it will take for player 1 to win the game. We shall extend this method of analysis for recursive games thusly. For a given recursive game $\Re^{\Gamma}$, we define p on terminal state γ to be

$$p(\gamma) = \begin{cases} 0 & \text{if } \gamma \text{ is a win for player } 1 \\ \infty & \text{otherwise} \end{cases} \tag{4.1}$$

Since this implies that player 1 is now the minimizing player, we shall define a new value function $\mathcal{V}'$ on two-player, zero-sum game Γ to be

$$\mathcal{V}'(\Gamma) = -\mathcal{V}((N, (C_1, C_2), \lambda x, y. - u(x, y))) \tag{4.2}$$

We can then define the value of the game to be a fixed point of a new function

$$T(\vec{v})[\gamma] = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ 1 + \mathcal{V}'(\mathcal{G}(\gamma, \vec{v})) & \text{otherwise} \end{cases} \tag{4.3}$$

Rather than representing an exact number of simultaneous rounds required for player 1 to achieve a win, the value of the game at a state γ is the expected number of rounds it will take player 1 to win the game playing forward from γ. Note that by using this definition for the value of the game, player 2 will value playing the game forever equally to winning.

It should quickly become apparent, though, that games with payoffs of ∞ are impossible to analyze due to the viral nature of infinity. Hence, we would rather define p as

$$p(\gamma) = \begin{cases} 0 & \text{if } \gamma \text{ is a win for player 1} \\ K & \text{otherwise} \end{cases} \tag{4.4}$$

for some arbitrary integer K so large that it may be considered a good approximation of infinity. Using arbitrarily large values in place of infinity in games is not a new concept [3], however our focus in doing so is different. While Başar and Olsder were concerned with games that are played for only a finite number of rounds and assumes that all games are played for K rounds, for us K is simply a very large number that the players may not know in advance. The players must play as if the game may never end, though player 1 will try to end the game by winning.

We can further normalize the analysis by introducing a *step factor* $\delta = \frac{1}{K}$ and using the following function to describe the value of the game

$$T(\vec{v})[\gamma] = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ min(1, \delta + \mathcal{V}'(\mathcal{G}(\gamma, \vec{v}))) & \text{otherwise} \end{cases} \tag{4.5}$$

If we also use p such that

$$p(\gamma) = \begin{cases} 0 & \text{if } \gamma \text{ is a win for player 1} \\ 1 & \text{otherwise} \end{cases} \tag{4.6}$$

we now get a game where the value at any state γ is in the range $[0, 1]$. We call these games *Games of Speed* - as the structure of such games encourages player 1 to hurry up and win, and conversely player 2 to do everything possible to slow player 1 down.

As it turns out, there is a formalization more closely related to the original solution proposed by Everett and constructed by Thuijsman and Vrieze. If we define p as

$$p(\gamma) = \begin{cases} 1 & \text{if } \gamma \text{ is a win for player 1} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

we can transform the game into a *Game of Survival* [8]. This formalization has the natural interpretation that the value of the game at any state γ is the probability that player 1 will win going forward from γ. Using the constructive algorithm from [37], we can find ϵ-strategies for such games, and since the value of the game at any state can never be smaller than 0 we don't even need to know the value of $\vec{v}$ for (3.14). This suggests that we should just pick the largest $\beta < 1$ possible and iterate (3.13) to convergence to get values for the game states.

4.1 Equivalence between Games of Speed and Games of Survival

As it turns out, in the limiting cases of $\beta = 1$ and $\delta = 0$, Games of Speed and Games of Survival are equivalent formalizations. To see this, we will make the strategy-preserving transformation of subtracting from 1 the value of the game at each state. This, of course, will change p in a game of speed to

$$p(\gamma) = \begin{cases} 1 & \text{if } \gamma \text{ is a win for player 1} \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

which is precisely the definition of p in a Game of Survival. This also gives us a new function for which to find a fixed point:

$$T'(\vec{v})[\gamma] = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ max(0, \mathcal{V}(\mathcal{G}(\gamma, \vec{v})) - \delta) & \text{otherwise} \end{cases} \tag{4.9}$$

To see why the second case of T' is as above, let $\vec{v}$ be a fixed point for T, and $\vec{v}'$ be the vector which results from subtracing each element of $\vec{v}$ from 1. Then for any $\gamma \in \Gamma \setminus \Gamma^T$,

$$\begin{aligned} \vec{v}'[\gamma] &= 1 - \vec{v}[\gamma] \\ &= 1 - T(\vec{v})[\gamma] \\ &= 1 - min(1, \delta + \mathcal{V}'(\mathcal{G}(\gamma, \vec{v}))) \\ &= max(0, 1 - (\delta + \mathcal{V}'(\mathcal{G}(\gamma, \vec{v})))) \\ &= max(0, 1 - \delta - \mathcal{V}'(\mathcal{G}(\gamma, \vec{v}))) \\ &= max(0, 1 - \mathcal{V}'(\mathcal{G}(\gamma, \vec{v})) - \delta) \\ &= max(0, 1 + \mathcal{V}(\mathcal{G}(\gamma, -\vec{v})) - \delta) \\ &= max(0, \mathcal{V}(\mathcal{G}(\gamma, \vec{1} - \vec{v})) - \delta) \\ &= max(0, \mathcal{V}(\mathcal{G}(\gamma, \vec{v}')) - \delta) \\ &= T'(\vec{v}')[\gamma] \end{aligned} \tag{4.10}$$

where $\vec{1}$ is the vector of size $|\Gamma|$ with all its elements set to 1.

Clearly analysis of the game of speed (4.5) and analysis of the corresponding game of survival (4.9) yield the same strategies. When this game of survival is compared to the discount factor-based game of survival (3.13), it is clear that they become the same game of survival as δ goes to 0 and β goes to 1. We expect to be able to find a constructive proof of the existence of ϵ-strategies for games of speed along the lines of [37]. This may prove useful as the subtraction in games of speed is computationally simpler than the multiplication in games of survival.

According to the results we obtain using an approach derived from [37], the best way to solve a game of survival is to use a β that is as close to 1 as possible, and the best way to solve a game of speed is to use the smallest δ possible. These quantities are limited by machine accuracy considerations; if β is too close to 1, roundoff errors will cause the computer to do the computation as if β were 1, and if δ is too close to 0, roundoff errors will cause the computer to do the computation as if delta were 0. In either case, the strategies that get extracted will not be suitable, as illustrated by the situation of the Guan sisters in Section 3.3. We will do an analysis, both theoretical and empirical, to find the most suitable values for β and δ based on numerical analysis techniques.

4.2 Game Theory of Lowered Expectations

In a two-player, zero-sum game, all Nash Equilibria are payoff-equivalent; so in theory a player should be indifferent as to which Equilibrium strategy to choose when more than one exists. However, there are cases where one Equilibrium strategy would be better than another in the case of an opponent making a mistake. Such strategies are called *subgame perfect*. Related to these situations are ones where it is impossible for a player to win when his/her opponent is playing optimally. As such, in this section we introduce the concept of *Game Theory of Lowered Expectations*, which is a theory to help a player make the best of a losing situation.

In an unwinnable situation, a player should still wish to choose strategies which in some sense push the state of the game closer to a winnable situation. As an example, we'll consider Right Turn from section 2.1. As is illustrated in Figure 1, there are only certain positions from which Thok is guaranteed to capture Lil' Cal. If he's not in one of

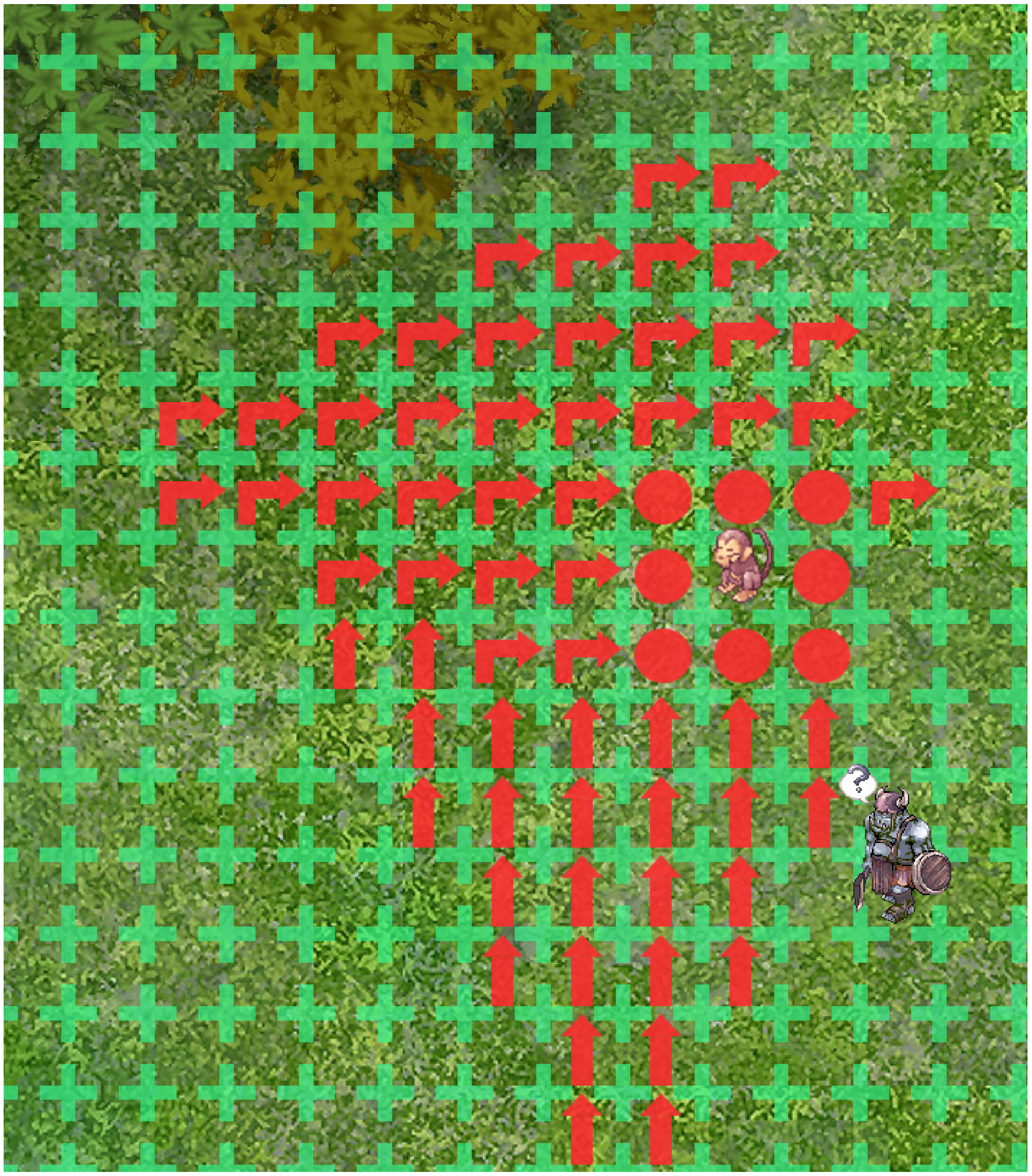

Figure 1: Policy for Thok shown graphically. The red arrows indicate what Thok should do when he finds himself in those squares. Squares without arrows indicate that nothing Thok does will lead to capture if Lil' Cal plays his cards right. The sprites representing Thok and Lil' Cal in this figure, along with the background graphic are from [16].

those squares, Lil' Cal has a strategy which enables him to always get away. However, it makes sense that Thok should want to stay as close to Lil' Cal as possible regardless of whether or not he's guaranteed to catch the monkey. Using just the definition of a Nash Equilibrium and the iterative process, Thok will be given no clue as to what he should do when not guaranteed a capture. On top of this, when Thok and Lil'Cal are further apart, Lil'Cal is given no clue as to what he should do either. The game would be more interesting, and more challenging for Lil'Cal, if he tried to stay as close as possible to Thok in these situations.

We propose three possibilities for solving this problem:

- α-expanded analysis
- α-expanded cooperative analysis
- β-expanded analysis

The names for these possibilities are tentative, necessary to allow us to talk about them. In all of these possibilities, the analysis explained in section 3.4 is done on the game, and the results of that are used in the subsequent "lowered expectations" analysis in an attempt to determine policies for states from which the game is unwinnable. Following is a brief descrpition of each possibility.

4.2.1 Alpha-Expanded Analysis

For a given game $\Re^\Gamma$, we will solve the game and obtain some $\vec{v_\beta}$. We will now build the *alpha expansion* of $\Re^\Gamma$, $\Re^\Gamma_\alpha = (\Gamma, \Gamma^T \cup \mathbb{X}, N, C^\Omega, C, f, p')$ where $\mathbb{X} = \{\gamma | \vec{v_\beta}[\gamma] \neq 0\}$ and

$$p'(\gamma) = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ 1 & \text{otherwise.} \end{cases} \tag{4.11}$$

Solve $\Re^\Gamma_\alpha$ and employ the resulting policy table.

We tried this during an analysis of Right turn; doing so made absolutely no difference in the resulting policy table. However, more analysis is necessary to determine whether or not this is the case in general.

4.2.2 Alpha-Expanded Cooperative Analysis

For a game $\Re^\Gamma$, obtain a policy for each non-terminal state in its alpha expansion $\Re^\Gamma_\alpha$ by having both players play to maximize the payoff to player 1. One way to do this is to find the largest payoff in the game matrix. Player 1 plays the row that value is in an player 2 plays the column it is in. Of course, if all the elements of the matrix are zero, this won't help, though the values might propagate to other game states.

4.2.3 Beta-Expanded Analysis

For a given game $\Re^\Gamma$, we will solve the game and obtain some $\vec{v_\beta}$. We will now build the *beta expansion* of $\Re^\Gamma$, $\Re^\Gamma_\beta = (\Gamma, \Gamma^T \cup \mathbb{X}, N, C^\Omega, C, f, p')$ where $\mathbb{X} = \{\gamma | (\exists c_1 \in C_1(\gamma))(\exists c_2 \in C_2(\gamma))[\vec{v_\beta}[f(\gamma, c_1, c_2)] \neq 0]\}$ and

$$p'(\gamma) = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ 1 & \text{otherwise.} \end{cases} \tag{4.12}$$

Solve $\Re^\Gamma_\beta$ and employ the resulting policy table.

5 Producing a Program from a Policy Table

One possible application of our approach is that of an offline learner. Using recursive game theory, we can obtain a program for playing the game in the form of a *policy table* - that is, a mapping from each state in the game to a strategy profile for that state. In practice, however, that table will be very large if at all finite. In this section, we propose a method for generating the policy table for a finite, tractible portion of a game's states to use as input to a classification algorithm. As an example, see Table 1 for a (small) examaple policy table for 1D1H1B as formalized in Section 3.5.

The analysis this method uses to derive the policy table is inspired by the analysis of Discrete Differential Games in [18]. We start from some terminal state (or possibly a collection of such) and progressively work backward, updating our value vector by resolving states as necessary.

Once we have a policy table, we'd like to use it to generate a program which is both more compact (the policy table, while finite and tractible, may still be incredibly large for non-trivial games) and more general (we'd like the AI to have some idea of what to do in states which don't show up in the table) while still being a recognizable program to a human. Classical AI has dealt heavily with inducing generalized programs from examples. The construction of such algorithms is outside the scope of the dissertation being proposed here. We will, however, run several of the leading ones on the policy tables our system produces and compare the results. The following systems will be considered:

1. OC1 [29]
2. C4.5/C5.0 [31]
3. Progol [28]
4. FOIDL [27]

OC1 and C5.0 are two leading decision tree induction systems, and Progol and FOIDL are two leading Inductive Logic Programming systems. Since we want the program to be something a human programmer can look at and understand, we reject methods such as neural networks and as such will not be discussing them.

6 Using forward Search with a heuristic evaluation function

Another possibility is to allow the game agent to use recursive game theory to make a decision on the fly about what to do next. Online planning has been studied in the literature. Examples include [4, 1, 5, 34, 39]. The system we propose in this section combines some of these available techniques with the theory of recursive games to make decisions as they are needed.

A recursive game is generated using Algorithm 5, then solved using the iterative process. The agent will then follow the strategy it obtains by computing the Nash Equilibrium for the current state using the value vector obtained from solving the game. The function h referenced in the algorithm is some *heuristic evaluation function* which attempts to approximiate the value of the game at its argument. This h will also be used to provide the values for the initial vector used in the iterative process. Plenty of work has been done in constructing heuristic evaluation functions for games [19, 7, 26, 21]. Discussion on how to derive such a function is outside the scope of this work, so we will just take for granted its existance. Ideally, this function should be admissible (in the classical sense).

Table 1: Example policy table for 1d1h1b

State	Policy
$(1,0,0,0)$	N/A
$(2,1,0,0)$	$((0,0,1),(0,0,1))$
$(3,1,1,0)$	$((1,0,0),(0,0,1))$
$(3,1,2,0)$	$((1,0,0),(0,0,1))$
$(3,1,3,0)$	$((1,0,0),(0,0,1))$
$(3,1,4,0)$	$((1,0,0),(0,0,1))$
$(3,1,5,0)$	$((1,0,0),(0,0,1))$
$(3,1,6,0)$	$((1,0,0),(0,0,1))$
$(3,1,7,0)$	$((1,0,0),(0,0,1))$
$(3,1,8,0)$	$((1,0,0),(0,0,1))$
$(3,1,9,0)$	$((1,0,0),(0,0,1))$
$(3,1,2,1)$	$((1,0,0),(1,0,0))$
$(3,1,3,2)$	$((1,0,0),(1,0,0))$
$(3,1,4,3)$	$((1,0,0),(1,0,0))$
$(3,1,5,4)$	$((1,0,0),(1,0,0))$
$(3,1,6,5)$	$((1,0,0),(1,0,0))$
$(3,1,7,6)$	$((1,0,0),(1,0,0))$
$(3,1,8,7)$	$((1,0,0),(1,0,0))$
$(3,1,9,8)$	$((1,0,0),(1,0,0))$

We conjecture that, for two distinct admissible heursitics h_1 and h_2, if h_1 dominates h_2 then a) the value vectors will converge to $\vec{v_\beta}$ more quickly if h_1 is used rather than h_2 for any β (and likewise for δ) and b) if the strategy profile σ_{h_1} resulting from using h_1 has a maximum state value error of ϵ_1 and the strategy profile σ_{h_2} resulting from using h_2 has a maximum state value error of ϵ_2, then $\epsilon_1 \leq \epsilon_2$.

Unlike the traditional online algorithms that are used for games such as Chess and Checkers, in this approach a graph is generated, not a tree. The heuristic evaluation function is applied to all the nodes in the graph and not just to the leaves of a tree. Iteration is needed to refine these values as one pass may not be enough, where conversely it is enough in the tree used for turn-based games.

We will run this algorithm on a subset of the games we use in Section 5. We will see how effectively a few iterations improve the played strategies compared to just using the heuristic evaluation function once on the states that are one round away from the current state and solving the resulting matrix game for the current state only. We will also give a high level recap of what happened during each game. Full play-by-plays will be inlcuded in the appendices.

7 Related Works

Despite the efforts of researchers such as Sue Epstein [11], who takes an approach based on Cognitive Science in developing her game player *Hoyle*, the most effective game playing programs so far are built on game theoretic principles rather than on imitating human cognitive processes (cite).

Work has been done in the Reinforcement Learning community, for example Gabor et al in [14] use an iterative learning process to develop strategies for multi-state games.

Algorithm 5 Making a recursive game on the fly. OverTimeLimit is abstract, its implementation will place an upper bound on how long genGame can execute.

```
function GENGAME($\gamma$)
    $Q \leftarrow []$
    $T \leftarrow \{\}$
    $C \leftarrow \{\}$
    $Push(Q, \gamma)$

    while $\neg Empty(Q) \wedge \neg OverTimeLimit$ do
        $curr \leftarrow Pop(Q)$

        for all $\alpha \in P1Moves$ do
            for all $\beta \in P2Moves$ do
                $succ \leftarrow f(curr, \alpha, \beta)$

                if $succ \in \Gamma^T$ then
                    $T \leftarrow T \cup \{succ\}$
                else if $\neg Contains(Q, succ)$ then
                    $Push(Q, succ)$
                end if
            end for
        end for

        $C \leftarrow C \cup \{curr\}$
    end while

    while $\neg Empty(Q)$ do
        $curr \leftarrow Pop(Q)$
        $T \leftarrow T \cup \{curr\}$
    end while

    return $(C \cup T, T, N, C^{\Omega}, C, f, \lambda x.(p(x)$ if $x \in \Gamma^T$ else $h(x)))$
end function
```

However, such methods tend to be very slow since they have to play many games against an opponent and a state will be visited no more than once in each game; in fact, only a small number of states are visited in each game, so it will take a long time for the learned game state values to converge to the best values.

Traditional planning based on STRIPS [13] and the like in most cases completely avoids an environment which changes in ways beyond the control of the planning agent. Even in more recent works such as [34], only changes via force of nature are considered - there are no other intelligent decision makers changing the environment, let alone deliberately antagonist ones.

More recently, multi-agent planning systems have dealt with planning in adversarial situations - however, application of recursive game theory in these systems is notably absent [9].

Daniel Andersson deals with turn-based recursive games [2], going as far as showing an almost linear-time algorithm for constructing winning strategies. He does not, however, deal with games of simultaneous movement. The few other papers applying recursive game theory to games also only deal with turn based games [25].

Every year, AAAI hosts a General Game Playing competition [15]. The goal is to develop an artificial intelligence which, once given the rules of a game, will play the game. The class of games used in the competition includes those which have simultaneous movement. While there are many different approaches to constructing general game players [19, 7, 33, 35, 26, 21, 20, 4], use of recursive game theory may be completely absent.

8 Roadmap

Going forth, we intend to accomplish several goals. We will:

- Expand Section 2 with formalizations for several other games which we'll use to test the software described by Sections 5 and 6.
- Present our own constructive proof on the existence of ϵ-strategies which will be more practical to use in a software system.
- Attempt to prove the same results about Games of Speed as Everett and Thuijsman and Vrieze did for recursive games using discount factors. This entails showing that, for every ϵ, there is some δ such that an iterative process using (4.5) will converge to a $\vec{v_\delta}$ such that using the values in $\vec{v_\delta}$ to compute a Nash equilibrium at any state γ will give us an ϵ-strategy profile for γ. It also involves showing that we'd like the smallest $\delta > 0$ we can choose when running the iterative process in code.
- Determine, given any $n \in \mathbb{N}$, the bounds for β and δ for an n-bit machine such that the choice of value does not exacerbate rounding errors. Intuitively, the hardware limit for δ would seem to be 2^{-n}. We would like to prove this lower bound, and to derive and prove an analagous upper-bound on β.
- Develop the software systems outlined in Sections 5 and 6 of this proposal. Both sections will be expanded to include technical documentation and formal descriptions of the core algorithms for their respective systems. We will also present and discuss the results of testing these systems on the games described in Section 2.

References

[1] Philip E. Agre and David Chapman, *Pengi: an implementation of a theory of activity*, Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87) (Kenneth D. Forbus and Howard E. Shrobe, eds.), Morgan Kaufmann, 1987, pp. 268–272.

[2] Daniel Andersson, Kristoffer A. Hansen, Peter B. Miltersen, and Troels B. Sørensen, *Simple Recursive Games*.

[3] T. Basar and G. J. Olsder, *Dynamic Noncooperative Game Theory, 2nd Ed.*, Academic, London, 1995.

[4] Y. Bjornsson and H. Finnsson, *CadiaPlayer: A Simulation-Based General Game Player*, Computational Intelligence and AI in Games, IEEE Transactions on **1** (2009), no. 1, 4–15.

[5] Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu, *Deep Blue*, Artificial Intelligence **134** (2002), no. 1-2, 57–83.

[6] Daniel Chester, *Review of Agent Cooperation within Adversarial Teams in Dynamic Environment Key Issues and Development Trends*, http://www.computingreviews.com/review/review_review.cfm?review_id=140684, November 2012.

[7] James Clune, *Heuristic evaluation functions for general game playing*, Ph.D. thesis, University of California Los Angeles, 2008.

[8] V. K. Domanskii, *Game of survival*, http://www.encyclopediaofmath.org/index.php/Game_of_survival.

[9] BartłomiejJózef Dzieńkowski and Urszula Markowska-Kaczmar, *Agent Cooperation within Adversarial Teams in Dynamic Environment Key Issues and Development Trends*, Transactions on Computational Collective Intelligence VI (NgocThanh Nguyen, ed.), Lecture Notes in Computer Science, vol. 7190, Springer Berlin Heidelberg, 2012, pp. 146–169.

[10] Entertainment Software Rating Board, *Video Game Industry Statistics*, http://www.esrb.org/about/video-game-industry-statistics.jsp, 2013.

[11] S. Epstein, *For the right reasons: The FORR architecture for learning in a skill domain*, Cognitive Science **18** (1994), no. 3, 479–511.

[12] Hugh Everett, *Recursive games*, Annals of Mathematics Studies, vol. 3, pp. 67–78, Princeton University Press, 41 William Street, Princeton, New Jersey, USA, 08540-5237, 1957.

[13] Richard E. Fikes and Nils J. Nilsson, *STRIPS: A new approach to the application of theorem proving to problem solving*, Artificial Intelligence **2** (1971), no. 3-4, 189–208.

[14] Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári, *Multi-criteria Reinforcement Learning*, Proceedings of the Fifteenth International Conference on Machine Learning (San Francisco, CA, USA), ICML '98, Morgan Kaufmann Publishers Inc., 1998, pp. 197–205.

[15] Michael Genesereth and Nathaniel Love, *General game playing: Overview of the AAAI competition*, AI Magazine **26** (2005), 62–72.

[16] Gravity Co., Ltd. and Lee Myoungjin, *Ragnarok Online*, [MMORPG] http://www.ragnarokonline.com/, 2003.

[17] Andrew Ilachinski, *Cellular Automata: A Discrete Universe*, World Scientific, Singapore, 2001.

[18] Rufus Isaacs, *Differential Games*, Wiley, Dover, Mineola, NY, 1965.

[19] David M. Kaiser, *Automatic feature extraction for autonomous general game playing agents*, Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems (New York, NY, USA), AAMAS '07, ACM, 2007.

[20] Tomoyuki Kaneko, Kazunori Yamaguchi, and Satoru Kawai, *Abstract Automatic Feature Construction and Optimization for General Game Player*.

[21] Gregory Kuhlmann, Kurt Dresner, and Peter Stone, *Automatic heuristic construction in a complete general game player*, In Proceedings of the Twenty-First National Conference on Artificial Intelligence, 2006.

[22] D. N. L. Levy, H. J. Berliner, and E. O. Thorpe, *Computer Games I*, Computer Games, Ishi Press, 2009.

[23] D. N. L. Levy, B. Wilcox, and E. O. Thorpe, *Computer Games II*, Computer Games, Ishi Press, 2009.

[24] R. Duncan Luce and Howard Raiffa, *Games and Decisions: Introduction and Critical Survey*, Dover Publications, Dover, NY, 1957.

[25] Peter B. Miltersen, *A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament*, In International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS, 2007.

[26] Makoto Miwa, Daisaku Yokoyama, and Takashi Chikayama, *Automatic Generation of Evaluation Features for Computer Game Players(Evaluation Function, Game Programming)*, Transactions of Information Processing Society of Japan **48** (2007), no. 11, 3428–3437.

[27] R. J. Mooney and M. E. Califf, *Induction of First-Order Decision Lists: Results on Learning the Past Tense of English Verbs*, Proceedings of the 5th International Workshop on Inductive Logic Programming (L. De Raedt, ed.), Department of Computer Science, Katholieke Universiteit Leuven, 1995, pp. 145–146.

[28] Stephen Muggleton, Wolfson Building, and Parks Road, *Inverse entailment and Progol*, 1995.

[29] Sreerama K. Murthy, Simon Kasif, and Steven Salzberg, *A System for Induction of Oblique Decision Trees*, Journal of Artificial Intelligence Research **2** (1994), 1–32.

[30] Roger B. Myerson, *Game theory: analysis of conflict*, Harvard University Press, 1997.

[31] J. Ross Quinlan, *C4.5: programs for machine learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[32] Jonathan Schaeffer, Neil Burch, Yngvi Bjornsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Paul Lu, and Steve Sutphen, *Checkers Is Solved*, Science (2007), 1144079+.

[33] Stephan Schiffel and Michael Thielscher, *M.: Fluxplayer: A successful general game player*, In: Proceedings of the AAAI National Conference on Artificial Intelligence, AAAI Press, 2007, pp. 1191–1196.

[34] Marcel J. Schoppers, *Representation and automatic synthesis of reaction plans*, Ph.D. thesis, Champaign, IL, USA, 1989.

[35] Michael Thielscher, *Answer Set Programming for Single-Player Games in General Game Playing*, Proceedings of the 25th International Conference on Logic Programming (Berlin, Heidelberg), ICLP '09, Springer-Verlag, 2009, pp. 327–341.

[36] L. C. Thomas, *Games, Theory and Applications*, Dover Books on Mathematics, Dover Publications, 1984.

[37] Thuijsman, *Note on recursive games*, Game Theory and Economic Applications (1992), 133–145.

[38] Ben G. Weber, Michael Mateas, and Arnav Jhala, *Building Human-Level AI for Real-Time Strategy Games*, Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems (San Francisco, California), AAAI Press, AAAI Press, 2011.

[39] Marco Wijdeven, *Game Trees in Realtime Games*, http://ai-depot.com/GameAI/GameTree.html, 2012.

[40] Wayne L. Winston and Munirpallam Venkataramanan, *Introduction to Mathematical Programming*, Brooks/Cole, 2003.