CONSTRUCTING STRATEGIES FOR GAMES WITH SIMULTANEOUS MOVEMENT

by

Jeremy Keffer

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Summer 2015

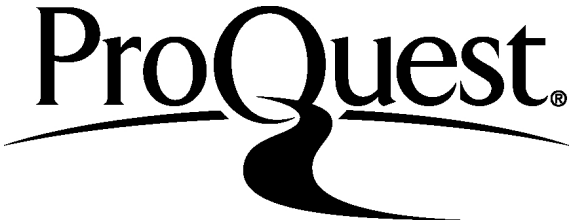
© 2015 Jeremy Keffer
All Rights Reserved

ProQuest Number: 3730247

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.



ProQuest 3730247

Published by ProQuest LLC (2015). Copyright of the Dissertation is held by the Author.

All rights reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

CONSTRUCTING STRATEGIES FOR GAMES WITH SIMULTANEOUS MOVEMENT

by

Jeremy Keffer

Approved: _____
Errol Lloyd, Ph.D.
Chair of the Department of Computer and Information Sciences

Approved: _____
Babatunde Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: _____
James G. Richards, Ph.D.
Vice Provost for Graduate and Professional Education

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Daniel L Chester, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

John Case, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Kathleen McCoy, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Jeffrey Heinz, Ph.D
Member of dissertation committee

ACKNOWLEDGMENTS

I've made lots of friends during my time at UD. I would love to acknowledge all of you, but the list is so long and I'm also afraid I'd leave someone out; so I'm going to keep this short.

I'd like to thank my wife, Shuwei. She spent the entirety of my graduate education living paycheck to paycheck with me. She put up with me every day. She was there when I was floundering, and when I wasn't sure I could ever finish a Ph.D. I know the experience was quite stressful for her.

I'd also like to thank Dan Chester. It is no exaggeration to say that without him, this dissertation would not have been possible. He went above and beyond what should be expected of a doctoral advisor. He gave me more help, patience, and kindness than I ever deserved. He is the true mastermind behind this work. It has been my greatest honor and privilege to play the John Watson to his Sherlock Holmes; and like Dr. Watson, I will now narrate this story.

TABLE OF CONTENTS

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

From the early days of AI, computers have been programmed to play games against human players. Most of the AI work has sought to build world-champion programs to play turn-based games such as Chess and Checkers, however computer games increasingly provide for entertaining real-time play. In this dissertation, we present an extension of recursive game theory, which can be used to analyze games involving simultaneous movement. We include an algorithm which can be used to practically solve recursive games, and present a proof of its correctness. We also define a game theory of lowered expectations to deal with situations where game theory currently fails to give players a definitive strategy, and demonstrate its applicability using several example games.

Chapter 1

INTRODUCTION

From the early days of AI, computers have been programmed to play games against human players [26, 27]. That continues to this day [46, 40, 22, 9, 41, 29, 43, 30, 5, 47]. Most of the AI work has sought to build world-champion programs to play turn-based games such as Chess and Checkers [6, 40]. But since its 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 [13]. 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*. Furthermore, these computer games, while challenging, are also meant to be entertaining, which is to say enjoyable, interesting.

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 [12, 8].

In this dissertation we examine a branch of *recursive game theory* which can be used to analyze computer games. Recursive game theory[15, 36, 45, 7] 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. This theory gives us an idea of how a good AI can play, and as such we'd like to know what it has to say before designing new computer game AIs. We also take it one step further by defining the *game theory of lowered expectations*, which allows a player to determine strategies in situations where traditional game theory would have nothing to say.

In order for the reader to understand our contributions to recursive game theory we must first provide some background on the theory and on game theory in general. We also need some example games to informally illustrate the kinds of games we're interested in, as well as illustrate various points of our theory. The rest of this dissertation is structured as follows: In Chapter 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 Chapter 3 we briefly discuss the relevant aspects of game theory in general and recursive game theory specifically required to understand our contributions to the theory. We then describe our contributions to recursive game theory in 4 and 5. In Chapter 4, we introduce a class of recursive games we call *Games of Speed* and show how there is a straightforward transformation between this class of games and another already known class of games. We then describe an algorithm for finding good strategies for both players of a recursive game, along with a proof of its correctness. In Chapter 5, we define and analyze game theory

of lowered expectations. Finally, in Chapter 6 we suggest several paths forward building on the work of this dissertation, as well as offer our closing remarks.

Chapter 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 [20]. Right Turn 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, the orc should prefer some actions over others. Consideration of these preferences leads to our *Game Theory of Lowered Expectations* in Chapter 5. 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[20].)

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:

$$(x, y) \ni |x| \leq 1 \wedge |y| \leq 1 \tag{2.1.1}$$

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* [19].

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 2.1.

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 forward. 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 2.1 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, and in Section 2.2.3 we describe a more complex instance which more closely resembles games that humans play.

2.2.1 Premise

In an extortion 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 successfully guard their precious mind-altering substance from outside looters? In the following chapters, we will suggest

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

how game theory, machine learning, and heuristic search can be used 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:

$$(u, v, 0, y) \ni u, v > 0 \wedge y \neq 0 \tag{2.2.1}$$

$$(u, 0, x, y) \ni u > 0 \tag{2.2.2}$$

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

$$(0, v, x, y) \tag{2.2.3}$$

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

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

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 2.2. In English, moving left or right will cause your location

Algorithm 2.2 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')

to decrement or increment, respectively. Attacking when your opponent

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

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.

2.2.3 2D Heroes vs. Bears

In this section, we introduce the rules of a variation of Bear Brawl we refer to as *Generalized Two-Dimensional Bear Brawl* (G2DBB).

This variation is played on a (possibly infinite) two-dimensional grid with both axes indexed by $\mathbb{N}$. A game of G2DBB can have any number of *Units*, each of which is either a *Hero* or a *Bear*. On each round of the game, a unit can either *Move* or *Attack* in one of the orthogonal directions. On each round, the hero units (as a team) decide upon their actions simultaneously with and independently from the bear units (who are also a team).

Formally, a unit u is a tuple $(s,(x,y),z) \ni s \in \{HERO, BEAR\} \wedge x,y,z \in \mathbb{N}$ We use $\mathbb{U}$ to denote the set of all possible Units. A hero h is a unit such that $s(h) = HERO$, and likewise a bear b is a unit such that $s(b) = BEAR$. For any unit u, the pair $(x(u), y(u))$ denotes u's position on the board, and $z(u)$ denotes u's *hit points*.

An *action* in G2DBB is a tuple $(a,d) \ni a \in \{\texttt{Move}, \texttt{Attack}\} \wedge d \in \{\texttt{North}, \texttt{South}, \texttt{West}, \texttt{East}\}$. We use $\mathbb{A}$ to denote the set of all possible actions; and $\vec{\mathbb{A}}$ to denote the set of all vectors of actions indexed by $\mathbb{U}$.

A state S in G2DBB is a tuple (U,x,y) where $U \subset \mathbb{U} \neq \emptyset$ is the set of units in the game, and $(\forall u \in U)\neg(\exists v \in U)[x(u) = x(v) \wedge y(u) = y(v) \wedge u \neq v]$ (in English, no two distinct units may occupy the same location); and the pair $(x(S), y(S))$ denotes the location of the powder. We use $\mathbb{S}$ to denote the

entire state space for the game. The following class of states are terminal and are win states for the heroes:

$$\begin{aligned}(U, x, y) \ni & \quad (\forall u \in U \neq \emptyset)(s(u) = HERO) \\ \vee & \quad (\exists u \in U)[(x(u) = x) \wedge (y(u) = y) \wedge (s(u) = HERO)]\end{aligned} \tag{2.2.4}$$

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

$$(U, x, y) \ni \neg(\exists u \in U)(s(u) = HERO) \tag{2.2.5}$$

Informally, the heroes win either when one of them obtains the powder (a hero's location is the same as the powder's) or when there are no bears (as we'll later see, a unit is removed from the game when its hit points are reduced to 0). If at any point there are no heroes left, the bears win.

The *Successor Function* $F : \mathbb{S} \times \vec{\mathbb{A}} \mapsto \mathbb{S}$ is defined in Algorithm 2.5. In English, moves are resolved first. A unit can move to its desired square iff that square is not inhabited by a unit who won't/can't move this round and there is not another unit who wishes to move to that square on this round. After all units who can and wish to move have, attacks are resolved. When a unit attacks a square, if a unit on the opposing team is occupying that square, the occupying unit loses one hit point. If a unit's HP drops to 0, then that unit dies and is removed from the game.

Algorithms 2.3 and 2.4 were extracted from Algorithm 2.5 for purposes of clarity. Algorithm 2.3 gives the location unit u intends to be in the following round given that u's action is a. Algorithm 2.4 determines whether or not a unit u intends to move and is not blocked from doing so by any other unit in U given that the actions of the units in U are $\vec{a}$.

Algorithm 2.3 INTENDED : $\mathbb{U} \times \mathbb{A} \mapsto \mathbb{N} \times \mathbb{N}$

function INTENDED(u, a)
  if $a = (\texttt{Move}, \texttt{North})$ **then**
    return $(x(u), y(u) + 1)$
  else if $a = (\texttt{Move}, \texttt{South})$ **then**
    return $(x(u), y(u) - 1)$
  else if $a = (\texttt{Move}, \texttt{East})$ **then**
    return $(x(u) + 1, y(u))$
  else if $a = (\texttt{Move}, \texttt{West})$ **then**
    return $(x(u) - 1, y(u))$
  else
    return $(x(u), y(u))$
  end if
end function

Algorithm 2.4 IMMOBILE : $\mathcal{P}(\mathbb{U}) \times \mathbb{U} \times \vec{\mathbb{A}} \mapsto \{\texttt{False}, \texttt{True}\}$

function IMMOBILE($U, u, \vec{a}$)
  if $a(\vec{a}[u]) \neq \texttt{Move}$ **then**
    return `True`
  else if $(\exists v \in U)[\text{INTENDED}(u, \vec{a}[u]) = \text{INTENDED}(v, \vec{a}[v]) \wedge v \neq u]$ **then**
    return `True`
  else if $(\exists v \in U)[\text{INTENDED}(u, \vec{a}[u]) = (x(v), y(v))]$ **then**
    return $\text{IMMOBILE}(U \setminus \{u\}, v, \vec{a})$
  else
    return `False`
  end if
end function

Algorithm 2.5 Successor Function

function F($S, \vec{a}$)
  $U' \leftarrow \emptyset$

  for all $u \in U(S)$ **do**
    if IMMOBILE$(U(S), u, \vec{a})$ **then**
      $U' \leftarrow U' \cup \{u\}$
    else
      $U' \leftarrow U' \cup \{(s(u), \text{INTENDED}(u, \vec{a}[u]), z(u))\}$
    end if
  end for

  for all $u \in U'$ **do**
    if $a(\vec{a}[u]) = \texttt{Attack}$ **then**
      $x' \leftarrow x(u)$
      $y' \leftarrow y(u)$
      if $d(\vec{a}[u]) = \texttt{North}$ **then**
        $y' \leftarrow y' + 1$
      else if $d(\vec{a}[u]) = \texttt{South}$ **then**
        $y' \leftarrow y' - 1$
      else if $d(\vec{a}[u]) = \texttt{East}$ **then**
        $x' \leftarrow x' + 1$
      else if $d(\vec{a}[u]) = \texttt{West}$ **then**
        $x' \leftarrow x' - 1$
      end if

      if $(\exists v \in U')[(x(v), y(v)) = (x', y')) \wedge (s(u) \neq s(v))]$ **then**
        $U' \leftarrow U' \setminus \{v\} \cup \{(s(v), (x(v), y(v)), z(v) - 1)\}$
      end if
    end if
  end for

  for all $u \in U'$ **do**
    if $z(u) \leq 0$ **then**
      $U' \leftarrow U' \setminus \{u\}$
    end if
  end for

  return $(U', x(S), y(S))$
end function

Figure 2.1: 2D Hero vs. Bears
One instance of the game we will use as an example involves one hero (Lady Eve) vs. three bears. Being outnumbered, Eve will begin the game with more HP than the bears. The starting location of Eve and those of the bears are arbitrary, as is the location of the powder. In this figure, the blue bear is at the same location as the powder.

Chapter 3

TWO-PLAYER, ZERO-SUM GAMES

In this section we will describe and discuss the background necessary in understanding our approach to games. We will introduce the concept of a two-player, zero-sum game and describe the formal framework which we use to analyze and discuss this class of games. We begin by stating the attributes which make a game a two-player, zero-sum game. Then, in Section 3.1 we define *matrix games* and describe necessary foundational theory in two-player zero-sum games. We then introduce the concept of a *recursive game* in Section 3.3, which is the formalization of two-player zero-sum games we use in our approach to games.

A *two-player, zero-sum game* is any game which satisfies the following properties:

1. There are two (and only two) opposing factions (players) making decisions which affect the outcome of the game.

2. At some point, the game is expected to terminate.

3. When the game ends, the two players are awarded a payoff. One player's gain is the other player's loss - *i.e.* The sum of the players' payoffs is 0.

Keeping these properties in mind, we will spend the remainder of this chapter presenting a formal definition of two-player, zero-sum games.

3.1 Matrix Games

The theory in this section is derived from [34]. It deals with the simplest kind of game, where the game is over after one round of play. Most treatments of game theory start with this kind of game. For a more in depth discussion of game theory and matrix games see [34], [44] or [28].

A *matrix game* Γ is an object with the structure:

$$\Gamma = (N, C, u) \tag{3.1.1}$$

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*.[1]

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 ambidextrous (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 Sven's 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 Sven's 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.

---

[1] 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 matrix 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 (3.1.2).

$$\left[\begin{array}{r|rr} u & DL & DR \\ \hline PL & 1 & -1 \\ PR & -1 & 1 \end{array}\right] \tag{3.1.2}$$

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 Matrix Games

A *strategy* σ_i for player i is defined to be a probability distribution over C_i - that is, $\sigma_i : C_i \mapsto \{x \in \mathbb{R} | 0 \leq x \leq 1\}$ 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 matrix game Γ, Let $\sigma = (\sigma_1, \sigma_2)$, where σ_1 and σ_2 are strategies for players 1 and 2, respectively, be a *strategy profile*. The *utility* $u'(\sigma)$ of σ is defined to be:

$$\sum_{c_1 \in C_1} \sum_{c_2 \in C_2} \sigma_1(c_1) \cdot \sigma_2(c_2) \cdot u(c_1, c_2) \tag{3.2.1}$$

A strategy profile σ is a *Nash equilibrium*[35] iff

$$(\forall \tau \in \Delta(C_1))[u'(\sigma) \geq u'((\tau, \sigma_2))] \wedge (\forall \tau \in \Delta(C_2))[u'(\sigma) \leq u'((\sigma_1, \tau))] \tag{3.2.2}$$

That is, neither player could increase her/his expected payoff by unilaterally deviating from her/his strategy as specified by σ.[2]

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

$$\begin{aligned} &\text{maximize } v \text{ subject to:} \\ &\qquad (\forall c_2 \in C_2) \left(v \leq \sum_{c_1 \in C_1} u(c_1, c_2) \cdot x_{c_1} \right) \\ &\qquad \sum_{c_1 \in C_1} x_{c_1} = 1 \\ &\qquad (\forall c_1 \in C_1)(x_{c_1} \geq 0) \end{aligned} \tag{3.2.3}$$

[2] Note that equation (3.2.2) 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 (3.2.2) must be reversed.

and

$$\begin{aligned}
&\text{minimize } w \text{ subject to:} \\
&(\forall c_1 \in C_1)\left(w \geq \sum_{c_2 \in C_2} u(c_1, c_2) \cdot y_{c_2}\right) \\
&\sum_{c_2 \in C_2} y_{c_2} = 1 \\
&(\forall c_2 \in C_2)(y_{c_2} \geq 0)
\end{aligned} \tag{3.2.4}$$

The resulting vectors $(x_{c_1}|c_1 \in C_1)$ of (3.2.3) and $(y_{c_2}|c_2 \in C_2)$ of (3.2.4) 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 matrix games see [49]. A very good discussion on matrix games as linear programming problems is also in [44].

The fundamental theorem about about two-person zero-sum matrix 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 [4, 28, 34, 44]. 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 between their possible actions, and the value of the game is 0.

3.3 Recursive Games

Recursive games were originally postulated by Everett in [15].

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

$$\Re^\Gamma = (\Gamma, \Gamma^T, N, C^\Omega, C, f, p) \tag{3.3.1}$$

Where N is as defined in equation (3.1.1); Γ is a (possibly infinite) 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 satisfies the property $(\forall \gamma \in \Gamma \setminus \Gamma^T)(\forall c_1 \in C_1(\gamma))(\forall c_2 \in C_2(\gamma))(f(\gamma, c_1, c_2) \in \Gamma)$;[3] 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.[4] 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 rogues 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).[5]

---

[3] In Everett's original formulation, the transition function mapped to a probability distribution over next states. We wanted to study games that were as deterministic as possible, so we simplified the definition of recursive game such that there is only one next state for each choice of the players.

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

[5] 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 (3.3.2) defines f.

$$\left[\begin{array}{c|cc} S & W & F \\ \hline W & S & X \\ R & X & D \end{array}\right] \tag{3.3.2}$$

p is defined in (3.3.3).

$$p(x) = \begin{cases} 1 & \text{if } x = X \\ -1 & \text{if } x = D \end{cases} \tag{3.3.3}$$

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 (3.3.1), 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 matrix game $\mathcal{G}(\gamma, \vec{v})$ to be

$$\mathcal{G}(\gamma, \vec{v}) = (N, (C_1(\gamma), C_2(\gamma)), \lambda x, y.\vec{v}[f(\gamma, x, y)]) \tag{3.4.1}$$

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,

$$F(\vec{v})[\gamma] = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ \mathcal{V}(\mathcal{G}(\gamma, \vec{v})) & \text{otherwise} \end{cases} \tag{3.4.2}$$

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

$$\vec{v} = F(\vec{v}) \tag{3.4.3}$$

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

In [15], 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 (3.4.3), 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 $\epsilon > 0$ there is a strategy profile σ^ϵ such that for all $\gamma \in \Gamma$, the expected value of the game at γ is within $\pm\epsilon$ of $\vec{v^*}[\gamma]$ 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 therefore neither woman wins (so the value of the game is effectively 0).

3.5 1D Hero vs. Bear as a Recursive Game

All of the games defined in Chapter 2 can be formalized as recursive games. As an example, 1D1H1B as a recursive game is:

$$(\Gamma, \Gamma^T, \{h, b\}, (\{L, R, A\}, \{L, R, A\}), (C, C), f, p) \tag{3.5.1}$$

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 (2.2.1), (2.2.2), and (2.2.3), C is a function which returns $\{L, R, A\}$ on any input, f is the function defined by algorithm 2.2, and the payoff function p is defined as such:

$$p(\gamma) = \begin{cases} 1 & \text{if } \gamma \text{ is a win for the hero} \\ -1 & \text{otherwise} \end{cases} \tag{3.5.2}$$

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

$$\vec{v}[(u, v, x, y)] = \begin{cases} 1 & \text{if } |x| < |y| \vee (|x| = |y| \wedge u > v) \vee (|x| > |y| \wedge u > v + 1) \\ -1 & \text{otherwise} \end{cases} \tag{3.5.3}$$

Possible policies for this game (in general, games can have more than one optimal strategy profile for each game state) are shown in Algorithms 3.1 and 3.2.

Algorithm 3.1 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** $x \geq y$ **else** L
  end if
end function

Algorithm 3.2 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** $y \geq x$ **else** L
  end if
end function

Chapter 4

GAMES OF SPEED AND SURVIVAL

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. [44, 47]

This goal suggests a type of game analysis that is inspired by Rufus Isaacs' differential game theory [20], 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

$$p(\gamma) = \begin{cases} 0 & \text{if } \gamma \text{ is a win for player 1} \\ \infty & \text{otherwise} \end{cases} \tag{4.0.1}$$

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

$$\mathcal{V}'(\Gamma) = -\mathcal{V}((N, (C_1, C_2), \lambda x, y. - u(x, y))) \tag{4.0.2}$$

This computes the minimized value of the matrix game for player 1 by first reversing the direction of payoffs, maximizing the payoff to player 1 in the usual manner, and then reversing the direction of payoff again. The vector of game values for states of the game will then be a fixed point of the function

$$T(\vec{v})[\gamma] = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ 1 + \mathcal{V}'(\mathcal{G}(\gamma, \vec{v})) & \text{otherwise} \end{cases} \tag{4.0.3}$$

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 difficult to analyze due to the way infinity propagates/dominates in arithmetic. Hence, we would rather define p as

$$p(\gamma) = \begin{cases} 0 & \text{if } \gamma \text{ is a win for player 1} \\ K & \text{otherwise} \end{cases} \tag{4.0.4}$$

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 [4]; 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 offset* $\delta = \frac{1}{K}$ and using the following function to describe the value of the game

$$T(\vec{v})[\gamma] = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ min(1, \delta + \mathcal{V}'(\mathcal{G}(\gamma, \vec{v}))) & \text{otherwise} \end{cases} \tag{4.0.5}$$

If we also use p such that

$$p(\gamma) = \begin{cases} 0 & \text{if } \gamma \text{ is a win for player 1} \\ 1 & \text{otherwise} \end{cases} \tag{4.0.6}$$

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.

Using a term to discount the expected value of the game based on how far the game is from a terminal state is nothing new. In [45], the authors introduced a $0 < \beta < 1$ term they called a *discount factor* which they used to scale values yielded by F. However, as we shall see in Section 4.2 the use of the δ step offset as a by product of defining Games of Speed is crucial to the algorithm for finding strategies for both players.

We would like a formalization more closely related to the original games studied by Everett, where player 1 is the maximizing player. There exists a class of games that has been studied in the literature called *Games*

of Survival (for player 2) [11]. The defining property of these games is the p function:

$$p(\gamma) = \begin{cases} 1 & \text{if } \gamma \text{ is a win for player 1} \\ 0 & \text{otherwise} \end{cases} \tag{4.0.7}$$

A Game of Survival 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.

4.1 Equivalence between Games of Speed and Games of Survival

As it turns out, games of speed and games of survival are equivalent formalizations.

Theorem 1. For every Game of Speed $\Re^{\Gamma}$, there is a corresponding Game of Survival $\Re^{\Gamma'}$.

Proof. We will first 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

$$p(\gamma) = \begin{cases} 1 & \text{if } \gamma \text{ is a win for player 1} \\ 0 & \text{otherwise} \end{cases} \tag{4.1.1}$$

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:

$$T'(\vec{v})[\gamma] = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ max(0, \mathcal{V}(\mathcal{G}(\gamma, \vec{v})) - \delta) & \text{otherwise} \end{cases} \tag{4.1.2}$$

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 subtracting each element of $\vec{v}$ from 1. Then for any $\gamma \in \Gamma \setminus \Gamma^T$,

$$\begin{aligned}
\vec{v}'[\gamma] &= 1 - \vec{v}[\gamma] \\
&= 1 - T(\vec{v})[\gamma] \\
&= 1 - min(1, \delta + \mathcal{V}'(\mathcal{G}(\gamma, \vec{v}))) \\
&= max(0, 1 - (\delta + \mathcal{V}'(\mathcal{G}(\gamma, \vec{v})))) \\
&= max(0, 1 - \delta - \mathcal{V}'(\mathcal{G}(\gamma, \vec{v}))) \\
&= max(0, 1 - \mathcal{V}'(\mathcal{G}(\gamma, \vec{v})) - \delta) \\
&= max(0, 1 + \mathcal{V}(\mathcal{G}(\gamma, -\vec{v})) - \delta) \\
&= max(0, \mathcal{V}(\mathcal{G}(\gamma, \vec{1} - \vec{v})) - \delta) \\
&= max(0, \mathcal{V}(\mathcal{G}(\gamma, \vec{v}')) - \delta) \\
&= T'(\vec{v}')[\gamma]
\end{aligned} \tag{4.1.3}$$

where $\vec{1}$ is the vector of size $|\Gamma|$ with all its elements set to 1. N.B. that several lines of (4.1.3) follow from the definitions of $\mathcal{G}$ in (3.4.1) and $\mathcal{V}'$ in (4.0.2). □

Clearly analysis of the game of speed (4.0.5) and analysis of the corresponding game of survival (4.1.2) yield the same strategies. When this step offset analysis of a game of survival is compared to the discount factor analysis from [45] of the same game, it is clear that the strategies obtained converge as δ goes to 0 and β goes to 1.

4.2 A Constructive Proof of the Existence of Epsilon Strategies

Most proofs of the existence of ϵ strategies require $\vec{v^*}$ to be known to get the strategy[15, 36, 45]. In [45], Thuijsman and Vrieze show a constructive proof of ϵ strategies, but this proof still relies upon the need to know the value of $\vec{v^*}$ to obtain strategies for the losing player. While Chatterjee et al do offer an algorithm for obtaining ϵ-strategies in [7], their

approach is not always sufficient for producing meaningful strategies in a computer gaming context - see Section 6.1 for a discussion of why this is the case.

In this section we present a constructive proof of ϵ strategies inspired by techniques used by Orkin in [36], and show that in practice it is possible to obtain strategies for both players without any a priori knowledge of $\vec{v^*}$.

To make our analysis easier, we make use of the *monus operator* $(\dot{-}) : \mathbb{R} \times \mathbb{R}$, where $x_1 \dot{-} x_2 = max(x_1 - x_2, 0)$ for any x_1, x_2. A useful fact about monus is:

Lemma 1.1. For any $n, x, y \in \mathbb{N}$, $(x \dot{-} ny) \dot{-} y = x \dot{-} (n+1)y$.

Proof.

$$\begin{aligned}
(x \dot{-} ny) \dot{-} y &= max(max(x - ny, 0) - y, 0) \\
&= max(max(x - (n+1)y, -y), 0) \\
&= max(x - (n+1)y, -y, 0) \\
&= max(x - (n+1)y, 0) \\
&= x \dot{-} (n+1)y
\end{aligned} \tag{4.2.1}$$

□

To further facilitate our analysis, we make use of the following definitions:

- For any $\Re^\Gamma$, we define the vector $\vec{v_0}$ thusly:

$$(\forall \gamma \in \Gamma)\vec{v_0}[\gamma] = \begin{cases} p(\gamma) & \text{if } \gamma \in \Gamma^T \\ 0 & \text{otherwise} \end{cases} \tag{4.2.2}$$

- For any function f whose range is a subset of its domain, we use $f^{\circ n}$ to be the *n-th iterate* of f. That is, $f^{\circ 0}(x) \stackrel{\text{def}}{=} x$ and $(\forall n > 0) f^{\circ n}(x) \stackrel{\text{def}}{=} (f \circ f^{\circ n-1})(x)$.
- For any $n \in \mathbb{N}$, $\vec{v_n} \stackrel{\text{def}}{=} F^{\circ n}(\vec{v_0})$.
- For any constant $c \in \mathbb{R}$, we define $\vec{c}$ to be the vector such that each component of $\vec{c}$ is c. The dimensionality of such a vector shall be inferred from its context.

- For any $\delta > 0$, we define $F_\delta(\vec{v})$ to be $F(\vec{v} \dot{-} \vec{\delta})$.
- For any $\delta > 0$, we define $\vec{v^\delta}$ to be $\lim_{n\to\infty} F_\delta^{\circ n}(\vec{v_0})$.
- For any $n \in \mathbb{N}$, $\vec{w_n} \stackrel{\text{def}}{=} F_\delta^{\circ n}(\vec{v_0})$. Here, δ is unbound. Its value will be obtained from the context where such $\vec{w_n}$ is used.[1]
- For any $\Re^\Gamma$, we let $\vec{\sigma_i}$ be a *strategy* for one round of play for player i. That is, for any $\gamma \in \Gamma$, $\vec{\sigma_i}[\gamma]$ denotes the strategy player i will play in state γ. If a player plays according to the same strategy in each round of a game, it is a *stationary strategy*.
- For any $\Re^\Gamma$, we define $Q(\vec{\sigma_1}, \vec{\sigma_2})$ to be a $|\Gamma|\times|\Gamma|$ matrix where $Q(\vec{\sigma_1}, \vec{\sigma_2})[i,j] = \sum_{(x,y)\in P_{ij}} (\vec{\sigma_1}[i](x) \cdot \vec{\sigma_2}[i](y))$ and $P_{ij} = \{(x,y) \mid f(i,x,y) = j\}$. That is, $Q(\vec{\sigma_1}, \vec{\sigma_2})$ is a matrix of probabilities where $Q(\vec{\sigma_1}, \vec{\sigma_2})[i,j]$ is the probability of transitioning to state j when in state i, player 1 plays $\vec{\sigma_1}$, and player 2 plays $\vec{\sigma_2}$. Note for $i \in \Gamma^T$ that P_{ij} is empty, so $Q(\vec{\sigma_1}, \vec{\sigma_2})[i,j] = 0$.

Theorem 2. For any recursive game $\Re^\Gamma$ with finite Γ and all terminal payoffs in the range $[0,1]$, the vector $\vec{v^*} \stackrel{\text{def}}{=} \lim_{n\to\infty} F^{\circ n}(\vec{v_0})$ is the *critical vector* as defined by Everett[15] and is the value of $\Re^\Gamma$. Further, for any ϵ, $\exists \delta_1 \in [0,1]$, $\exists k_1 \in \mathbb{N}$ such that component 1 of the output of Algorithm 4.1 is an ϵ-strategy for player 1, and $\exists \delta_2 \in [0,1]$, $k_2 \in \mathbb{N}$ such that component 2 of the output of Algorithm 4.1 is an ϵ-strategy for player 2 when the game's length has an upper bound of some n.

Algorithm 4.1 Procedure for constructing ϵ-strategies

```
function FIND_STRATEGY(ℜ^γ, δ, k)
    v⃗ ← v⃗_0

    for all i ∈ ℕ ∋ i < k do
        v⃗ ← F(v⃗ ∸ δ)
    end for

    return [𝒩(G(γ, v⃗)) | ∀γ ∈ Γ \ Γ^T]
end function
```

[1] **N.B.:** Undecorated $\vec{v}$ and $\vec{w}$ are frequently used to refer to arbitrary vectors which may be unrelated to any iteration of F or F_δ, respectively.

Proof. First, we'll define a partial ordering upon value vectors. For two vectors $\vec{v}$ and $\vec{w}$, we say that $\vec{v} \leq \vec{w}$ if and only if $\forall(i \in |\vec{v}|)(\vec{v}[i] \leq \vec{w}[i])$. Further, we say that $\vec{v} < \vec{w}$ if and only if $\vec{v} \leq \vec{w} \wedge \exists(i \in |\vec{v}|)(\vec{v}[i] < \vec{w}[i])$.

To continue the proof, Lemmas 2.1 thru 2.6 provide support for Lemma 2.7 (along with Corollaries 2.7.1 and 2.7.2), Lemma 2.8, and Lemma 2.9. Corollary 2.7.1 shows the existence of ϵ strategies for player 1, while Corollary 2.7.2 shows that Algorithm 4.1 will provide ϵ strategies for player 1. Likewise, Lemma 2.8 shows the existence of ϵ strategies for player 2, and Lemma 2.9 shows that Algorithm 4.1 will provide ϵ strategies for player 2 given that the game ends within n rounds, where n is some arbitrarily large integer.

Lemma 2.1. Given value vectors $\vec{v}$ and $\vec{w}$, $\vec{v} \leq \vec{w} \implies F(\vec{v}) \leq F(\vec{w})$

Proof. Let γ be an arbitrary state in $\Gamma(\Re^\Gamma)$. If $G(\gamma, \vec{v}) = G(\gamma, \vec{w})$, then $F(\vec{v})[\gamma] = F(\vec{w})[\gamma]$. Otherwise, let $\sigma^v = (\sigma_1^v, \sigma_2^v)$ be a Nash equilibrium for $G(\gamma, \vec{v})$, and likewise $\sigma^w = (\sigma_1^w, \sigma_2^w)$ be a Nash equilibrium for $G(\gamma, \vec{w})$. Then, $F(\vec{v})[\gamma] = u'(\sigma^v, G(\gamma, \vec{v})) \leq u'((\sigma_1^v, \sigma_2^w), G(\gamma, \vec{v})) \leq u'((\sigma_1^v, \sigma_2^w), G(\gamma, \vec{w})) \leq u'(\sigma^w, G(\gamma, \vec{w})) = F(\vec{w})[\gamma]$. The middle inequality follows from the fact that the value of a matrix game for a given combination of strategies for the two players increases by a non-negative value if the elements of the matrix are increased by non-negative values and the players do not change their strategies, and the other equalities and inequalities follow from the definition of Nash equilibrium. □

Corollary 2.1.1. $(\forall \delta > 0)(\vec{v^\delta} \leq \vec{v^*})$

Proof. We shall show that, for an arbitrary δ, $\forall(n \in \mathbb{N})(F_\delta^{\circ n}(\vec{v_0}) \leq F^{\circ n}(\vec{v_0}))$ By induction on n

- The case of $n = 0$ is trivially true.

- By the induction hypothesis, $F_\delta^{\circ n}(\vec{v_0}) \leq F^{\circ n}(\vec{v_0})$. By Lemma 2.1, $F_\delta^{\circ n+1}(\vec{v_0}) = F_\delta(F_\delta^{\circ n}(\vec{v_0})) = F(F_\delta^{\circ n}(\vec{v_0}) \dot{-} \vec{\delta}) \leq F(F_\delta^{\circ n}(\vec{v_0})) \leq F(F^{\circ n}(\vec{v_0})) = F^{\circ n+1}(\vec{v_0})$.

□

Corollary 2.1.2. $(\forall n \in \mathbb{N})(\vec{v}_n \leq \vec{v}_{n+1})$

Proof. By induction on n

- Since $\vec{v}_0$ is the smallest vector of game state values, $\vec{v}_0 \leq \vec{v}_1$ is trivially true.
- By the induction hypothesis, $\vec{v}_{n-1} \leq \vec{v}_n$. By Lemma 2.1, $\vec{v}_n = F(\vec{v}_{n-1}) \leq F(\vec{v}_n) = \vec{v}_{n+1}$.

□

Lemma 2.2. For any matrix game $\Gamma_1 = (N, C, u_1)$ with $\Gamma_2 = (N, C, u_2 = \lambda x, y.u_1(x, y) - \delta)$, $\mathcal{V}(\Gamma_2) = \mathcal{V}(\Gamma_1) - \delta$.

Proof. Given $\sigma = (\sigma_1, \sigma_2)$, it should be obvious that $u'(\sigma, \Gamma_2) = u'(\sigma, \Gamma_1) - \delta$. Let $\sigma = (\sigma_1, \sigma_2)$ be a Nash equilibrium for Γ_2. Assume that σ is not a Nash equilibrium for Γ_1. Assume without loss of generality that player 1 can unilaterally deviate and increase their payoff. Therefore, there exists a strategy τ for player 1 such that $u'((\tau, \sigma_2), \Gamma_1) > u'(\sigma, \Gamma_1)$. Thus we can say $u'((\tau, \sigma_2), \Gamma_2) = u'((\tau, \sigma_2), \Gamma_1) - \delta > u'(\sigma, \Gamma_1) - \delta = u'(\sigma, \Gamma_2)$, which is a contradiction. Therefore, σ must be a Nash equilibrium for Γ_1. Thus we can say that $\mathcal{V}(\Gamma_2) = u'(\sigma, \Gamma_2) = u'(\sigma, \Gamma_1) - \delta = \mathcal{V}(\Gamma_1) - \delta$. □

Lemma 2.3. $\vec{v^*}$ is a fixed point of F.

Proof. Let n be such that $\vec{v_n} > \vec{v^*} - \vec{\epsilon}$ for some arbitrary $\epsilon > 0$. Then, by Corollary 2.1.2 and Lemmas 2.1 and 2.2, $\vec{v^*} \geq F(\vec{v_n}) \geq F(\vec{v^*} - \vec{\epsilon}) \geq F(\vec{v^*}) - \vec{\epsilon}$, from which follows that $F(\vec{v^*}) \leq \vec{v^*} + \vec{\epsilon}$. Since ϵ can get arbitrarily close to 0, it must be the case that $F(\vec{v^*}) \leq \vec{v^*}$. Now assume that $F(\vec{v^*}) < \vec{v^*}$. Let i

be such that $v_i > F(\vec{v^*})$. By Lemma 2.1, $F(\vec{v_i}) \leq F(\vec{v^*})$. Thus we can say by Corollary 2.1.2 that $\vec{v_i} \leq F(\vec{v_i}) \leq F(\vec{v^*})$, which is a contradiction. Therefore, $\vec{v^*} = F(\vec{v^*})$. □

Lemma 2.4. $\vec{v^*}$ is the smallest non-negative fixed point of F.

Proof. Let $\vec{w} \geq \vec{0}$ be an arbitrary fixed point of F. We will now show by mathematical induction that $(\forall n \in \mathbb{N})[\vec{v_n} \leq \vec{w}]$.

- The base case is trivially true; since the values corresponding to terminal states are constant across all value vectors, and the value for every non-terminal in $\vec{v_0}$ is 0, which is the smallest possible value.
- By the induction hypothesis, $\vec{v_n} \leq \vec{w}$. By Lemma 2.1, $\vec{v_{n+1}} = F(\vec{v_n}) \leq F(\vec{w}) = \vec{w}$.

Therefore, $\vec{v^*} \leq \vec{w}$. □

By Lemmas 2.3 and 2.4, we assert that the limit $\vec{v^*}$ of iterating F is the critical vector for recursive game $\Re^\Gamma$ as defined by Everett [15].

Lemma 2.5. $(\forall i \geq 0)(\forall \delta > 0)(\vec{w_i} \geq \vec{v_i} - i \cdot \vec{\delta})$

Proof. Remember that $(\forall i)(\vec{w_i} \stackrel{\text{def}}{=} F_\delta^{\circ i}(\vec{v_0}))$.

Proposition 2.5.1. For any δ, and any $\vec{v}$, $F_\delta(\vec{v}) \geq F(\vec{v}) - \vec{\delta}$.

Proof. By Lemma 2.2 and the definition of F, we have that $F(\vec{v}) - \vec{\delta} \leq F(\vec{v} - \vec{\delta})$. By Lemma 2.1, we have $F(\vec{v} - \vec{\delta}) \leq F(\vec{v} \dot{-} \vec{\delta}) = F_\delta(\vec{v})$. □

Let γ be an arbitrary non-terminal state. Then, by induction on i:

- Consider the base case of iteration 1. In this case, by Proposition 2.5.1, $\vec{w_1}[\gamma] = F_\delta(\vec{v_0})[\gamma] \geq F(\vec{v_0})[\gamma] - \delta$. But $F(\vec{v_0})[\gamma] = \vec{v_1}[\gamma]$, so $\vec{w_1}[\gamma] \geq \vec{v_1}[\gamma] - 1 \cdot \delta$.
- By the induction hypothesis, $\vec{w_i}[\gamma] \geq \vec{v_i}[\gamma] - i \cdot \delta$. Then, $\vec{w}_{i+1}[\gamma] = F(\vec{w_i} \dot{-} \vec{\delta})[\gamma] \geq F(\vec{v_i} - i \cdot \vec{\delta} - \vec{\delta})[\gamma] = F(\vec{v_i} - (i+1) \cdot \vec{\delta})[\gamma]$ by Lemma 2.1 and the definition of $(\dot{-})$. Furthermore, $F(\vec{v_i} - (i+1) \cdot \vec{\delta})[\gamma] = F(\vec{v_i})[\gamma] - (i+1) \cdot \delta = \vec{v}_{i+1}[\gamma] - (i+1) \cdot \delta$ by Lemma 2.2. Therefore, $\vec{w}_{i+1}[\gamma] \geq \vec{v}_{i+1}[\gamma] - (i+1) \cdot \delta$.

This, plus the fact that $(\forall \gamma \in \Gamma^T)(\vec{v_i}[\gamma] = \vec{w_i}[\gamma])$ by the definition of F proves the lemma. □

Lemma 2.6. Given an $\epsilon > 0$, there exists $\delta > 0$ such that $\vec{v^\delta} \geq \vec{v^*} - \vec{\epsilon}$.

Proof. Let $i \in \mathbb{N}$ be such that $\vec{v_i} \geq \vec{v^*} - \frac{1}{2} \cdot \vec{\epsilon}$. Let δ be such that $i \cdot \delta \leq \frac{\epsilon}{2}$. Then, by Lemma 2.5 $F_\delta^{\circ i}(\vec{v_0})[\gamma] \geq \vec{v_i}[\gamma] - i \cdot \delta \geq \vec{v_i}[\gamma] - \frac{1}{2} \cdot \vec{\epsilon} \geq \vec{v^*} - \frac{1}{2} \cdot \vec{\epsilon} - \frac{1}{2} \cdot \vec{\epsilon} = \vec{v^*} - \vec{\epsilon}$. Since $\vec{v^\delta} \geq F_\delta^{\circ i}(\vec{v_0})$, it is also greater than or equal to $\vec{v^*} - \vec{\epsilon}$. □

It is worth pointing out that Corollary 2.1.1 together with Lemma 2.6 allow us to say that for any $\epsilon > 0$ there is a $\delta > 0$ such that $\vec{v^\delta}$ lies between $\vec{v^*} - \vec{\epsilon}$ and $\vec{v^*}$.

Lemma 2.7. For any ϵ, given any vector $\vec{w}$, $\delta > 0 \ni (\vec{w} \geq \vec{v^*} - \vec{\epsilon}) \wedge (F_\delta(\vec{w}) \geq \vec{w})$, the vector $\vec{\sigma_1} \ni (\forall \gamma \in \Gamma \setminus \Gamma^T)[\vec{\sigma_1}[\gamma] = \sigma_1(\mathcal{N}(G(\gamma, \vec{w} \dot{-} \vec{\delta})))]$ is an ϵ strategy for Player 1.

Proof. Let $\vec{\sigma_2} \ni (\forall \gamma \in \Gamma \setminus \Gamma^T)(\vec{\sigma_2}[\gamma] = \sigma_2(\mathcal{N}(G(\gamma, \vec{w} \dot{-} \vec{\delta})))$. Then we can say $F_\delta(\vec{w}) = \vec{v_0} + Q(\vec{\sigma_1}, \vec{\sigma_2})(\vec{w} \dot{-} \vec{\delta})$. Therefore $\vec{v_0} + Q(\vec{\sigma_1}.\vec{\sigma_2})(\vec{w} \dot{-} \vec{\delta}) \geq \vec{w}$. By definition of Nash equilibrium $\vec{v_0} + Q(\vec{\sigma_1}, \vec{\tau})(\vec{w} \dot{-} \vec{\delta}) \geq \vec{v_0} + Q(\vec{\sigma_1}, \vec{\sigma_2})(\vec{w} \dot{-} \vec{\delta})$ where $\vec{\tau}$ is some arbitrary strategy for player 2. Hence $\vec{v_0} + Q(\vec{\sigma_1}, \vec{\tau})(\vec{w} \dot{-} \vec{\delta}) \geq \vec{w}$ for any τ.

As such, $\vec{v_0} \geq \vec{w} - Q(\vec{\sigma_1}, \vec{\tau})(\vec{w} \dot{-} \vec{\delta})$. Let $\vec{\tau_1}, \vec{\tau_2}, \vec{\tau_3}, \ldots$ be an infinite sequence of strategies such that $\vec{\tau_i}$ is played by player 2 on the ith round of play, Q_0 be the identity matrix, and $(\forall i > 0)(Q_i = Q(\vec{\sigma_1}, \vec{\tau_i}))$. It should be clear that

the expected value of the game if these strategies are followed for up to n rounds is $\sum_{i=0}^{n}\prod_{j=0}^{i} Q_j\vec{v_0}$. Thus we can say:

$$\begin{aligned}
\sum_{i=0}^{n}\prod_{j=0}^{i} Q_j\vec{v_0} &\geq \sum_{i=0}^{n}\prod_{j=0}^{i} Q_j(\vec{w} - Q_{i+1}(\vec{w} \dot{-} \vec{\delta})) \\
&\geq \sum_{i=0}^{n}(\prod_{j=0}^{i} Q_j\vec{w} - \prod_{j=0}^{i} Q_jQ_{i+1}(\vec{w} \dot{-} \vec{\delta})) \\
&\geq \sum_{i=0}^{n}(\prod_{j=0}^{i} Q_j\vec{w} - \prod_{j=0}^{i+1} Q_j(\vec{w} \dot{-} \vec{\delta})) \\
&\geq \sum_{i=0}^{n}\prod_{j=0}^{i} Q_j\vec{w} - \sum_{i=0}^{n}\prod_{j=0}^{i+1} Q_j(\vec{w} \dot{-} \vec{\delta}) \\
&\geq \sum_{i=0}^{n}\prod_{j=0}^{i} Q_j\vec{w} - \sum_{i=1}^{n+1}\prod_{j=0}^{i} Q_j(\vec{w} \dot{-} \vec{\delta}) \\
&\geq \vec{w} + \sum_{i=1}^{n}\prod_{j=0}^{i} Q_j\vec{w} - \sum_{i=1}^{n}\prod_{j=0}^{i} Q_j(\vec{w} \dot{-} \vec{\delta}) - \prod_{j=0}^{n+1} Q_j(\vec{w} \dot{-} \vec{\delta}) \\
&\geq \vec{w} + \sum_{i=1}^{n}\prod_{j=0}^{i} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) - \prod_{j=0}^{n+1} Q_j(\vec{w} \dot{-} \vec{\delta}) \\
&\geq \vec{w} + \sum_{i=1}^{n}\prod_{j=0}^{i} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) - \eta\prod_{j=0}^{n+1} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta}))
\end{aligned} \tag{4.2.3}$$

Where

$$\eta = max(map(f, (\vec{w} \dot{-} \vec{\delta}), (\vec{w} - (\vec{w} \dot{-} \vec{\delta})))) \tag{4.2.4}$$

where

$$f(x,y) = \begin{cases} 0 & \text{if } y = 0 \\ \frac{x}{y} & \text{otherwise} \end{cases} \tag{4.2.5}$$

$max(\vec{v})$ is the *maximal component* of $\vec{v}$ - that is, $(\forall\gamma \in \Gamma)(\vec{v}[\gamma] \leq max(\vec{v}))$, and $(\forall\gamma \in \Gamma)(map(f, v, w)[\gamma] = f(v[\gamma], w[\gamma]))$. The last inequality in (4.2.3) is justified by the following proposition.

Proposition 2.7.1. $\eta \cdot (\vec{w} - (\vec{w} \dot{-} \vec{\delta})) \geq (\vec{w} \dot{-} \vec{\delta})$

Proof. Consider the following two cases:

1. $\eta > 0$: Let $\vec{x} = map(f, (\vec{w} \dot{-} \vec{\delta}), (\vec{w} - (\vec{w} \dot{-} \vec{\delta})))$, and γ be such that $\eta = \vec{x}[\gamma]$. It should be clear that $\eta \cdot (\vec{w} - (\vec{w} \dot{-} \vec{\delta}))[\gamma] = (\vec{w} \dot{-} \vec{\delta})[\gamma]$. Then, by definition of max, $(\forall g \in \Gamma)(\eta \cdot (\vec{w} - (\vec{w} \dot{-} \vec{\delta}))[g] \geq (\vec{w} \dot{-} \vec{\delta})[g]$.

2. $\eta = 0$: Then, $(\forall \gamma \in \Gamma)(map(f, (\vec{w} \dot{-} \vec{\delta}), (\vec{w} - (\vec{w} \dot{-} \vec{\delta})))[\gamma] = 0)$. As such, $(\forall \gamma \in \Gamma)(\vec{w} \dot{-} \vec{\delta} = 0)$. To see why this is true for $\gamma \ni (\vec{w} - (\vec{w} \dot{-} \vec{\delta}))[\gamma] = 0$, consider that $\delta > 0$ and that all components of $\vec{w}$ are at least 0. The only value for $\vec{w}[\gamma]$ which satisfies $(\vec{w} - (\vec{w} \dot{-} \vec{\delta}))[\gamma] = \vec{w}[\gamma] - (\vec{w}[\gamma] \dot{-} \delta) = 0$ is 0. Thus $\vec{w} \dot{-} \vec{\delta} = \vec{0}$. Hence, $\eta \cdot (\vec{w} - (\vec{w} \dot{-} \vec{\delta})) \geq \vec{w} \dot{-} \vec{\delta}$.

□

Proposition 2.7.2. $\lim_{n \to \infty} \prod_{j=0}^{n} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) = 0$

Proof. By (4.2.3), we have that

$$\begin{aligned} \sum_{i=0}^{n} \prod_{j=0}^{i} Q_j \vec{v_0} &\geq \vec{w} + \sum_{i=1}^{n} \prod_{j=0}^{i} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) - \eta \prod_{j=0}^{n+1} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) \\ \sum_{i=0}^{n} \prod_{j=0}^{i} Q_j \vec{v_0} - \vec{w} + \eta \prod_{j=0}^{n+1} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) &\geq \sum_{i=1}^{n} \prod_{j=0}^{i} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) \end{aligned} \tag{4.2.6}$$

Since the $\sum_{i=0}^{n} \prod_{j=0}^{i} Q_j \vec{v_0}$ term is the expected payoff after playing up to n rounds of the game, and that payoff cannot be more than 1, it is upper bounded by $\vec{1}$. Likewise, $\vec{w}$ and $\vec{w} - (\vec{w} \dot{-} \vec{\delta})$ are upper bounded by $\vec{1}$. Since the rows of the Q matrices are probability distributions, the product of Q with a vector upper bounded by $\vec{1}$ produces another vector upper bounded by $\vec{1}$. η cannot be larger than $\frac{1}{\delta}$. So the left side of the second inequality of (4.2.6) has an upper bound of $\frac{\delta+1}{\delta} \cdot \vec{1}$, which means that the sum on the right side of that inequality has the same upper bound. But that sum is a sum of positive vectors, So the last term in that sum must be tending toward $\vec{0}$ as n gets arbitrarily large. □

By Proposition 2.7.1, $(\sum_{i=1}^{n} \prod_{j=0}^{i} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) - \eta \prod_{j=0}^{n+1} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) \leq (\sum_{i=1}^{n} \prod_{j=0}^{i} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) - \prod_{j=0}^{n+1} Q_j(\vec{w} \dot{-} \vec{\delta}))$. To show that $(\exists n)(\sum_{i=1}^{n} \prod_{j=0}^{i} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) - \eta \prod_{j=0}^{n+1} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta})) \geq 0)$ and subsequently that player 1's expected payoff is greater than or equal to $\vec{v^*} - \vec{\epsilon}$, consider that by Proposition 2.7.2 $\eta \prod_{j=0}^{n} Q_j(\vec{w} - (\vec{w} \dot{-} \vec{\delta}))$ is getting arbitrarily close to 0 as $n \to \infty$, and is thus at

some point overtaken by $\sum_{i=1}^{n}\prod_{j=0}^{i}Q_j(\vec{w}-(\vec{w}\dot{-}\vec{\delta}))$. Hence $\sum_{i=0}^{n}\prod_{j=0}^{i}Q_j\vec{v_0}\geq\vec{w}\geq\vec{v^*}-\vec{\epsilon}$, which is sufficient to establish that σ_1 is an ϵ strategy for player 1. □

Corollary 2.7.1. For any $\epsilon>0$, there exists $\delta>0$ such that the vector $\vec{\sigma_1}\ni(\forall\gamma\in\Gamma\setminus\Gamma^T)[\vec{\sigma_1}[\gamma]=\sigma_1(\mathcal{N}(G(\gamma,\vec{v^\delta}\dot{-}\vec{\delta}))]$ is an ϵ strategy for Player 1.

Proof. By Lemmas 2.6 and 2.7, and by definition of F_δ. □

Corollary 2.7.2. For any $\epsilon>0$, there exists $\delta>0$ and $k\in\mathbb{N}$ such that the vector $\vec{\sigma_1}\ni(\forall\gamma\in\Gamma\setminus\Gamma^T)[\vec{\sigma_1}[\gamma]=\sigma_1(\mathcal{N}(G(\gamma,\vec{v_k}\dot{-}\vec{\delta})))]$ is an ϵ strategy for Player 1.

Proof. Given $\epsilon>0$, choose $\delta>0$ such that $\vec{v^\delta}\geq\vec{v^*}-\frac{1}{2}\cdot\vec{\epsilon}$, which exists by Lemma 2.6. Choose k such that $v_k=F_\delta^{\circ k}(\vec{v_0})\geq\vec{v^\delta}-\frac{1}{2}\cdot\vec{\epsilon}$, which exists by definition of $\vec{v^\delta}$. The corollary then follows by Lemmas 2.1 and 2.7. □

Lemma 2.8. For any $\epsilon>0$, given some $n\in\mathbb{N}$ which is an upper bound on the length (in rounds) of the game, there exists $\delta>0$ such that the vector $\vec{\sigma_2}\ni(\forall\gamma\in\Gamma\setminus\Gamma^T)[\vec{\sigma_2}[\gamma]=\sigma_2(\mathcal{N}(G(\gamma,\vec{v^\delta}\dot{-}\vec{\delta}))]$ is an ϵ strategy for Player 2.

Proof. Given $\epsilon>0$, by Lemma 2.6 there is a $\delta>0$ such that $\vec{v^\delta}\geq\vec{v^*}-\frac{1}{2n}\cdot\vec{\epsilon}$.

Let $\vec{\sigma_1}\ni(\forall\gamma\in\Gamma\setminus\Gamma^T)[\vec{\sigma_1}[\gamma]=\sigma_1(\mathcal{N}(G(\gamma,\vec{v^\delta}\dot{-}\vec{\delta}))]$. Then we can say $F_\delta(\vec{v^\delta})=\vec{v_0}+Q(\vec{\sigma_1},\vec{\sigma_2})(\vec{v^\delta}\dot{-}\vec{\delta})$. Therefore, by definition of Nash equilibrium $\vec{v_0}+Q(\vec{\tau},\vec{\sigma_2})(\vec{v^\delta}\dot{-}\vec{\delta})\leq\vec{v^\delta}$, where $\vec{\tau}$ is some arbitrary strategy for player 1. Since $\vec{v^\delta}<\vec{v^*}$, $\vec{v_0}+Q(\vec{\tau},\vec{\sigma_2})(\vec{v^\delta}\dot{-}\vec{\delta})\leq\vec{v^*}$.

As such, $\vec{v_0}\leq\vec{v^*}-Q(\vec{\tau},\vec{\sigma_2})(\vec{v^\delta}\dot{-}\vec{\delta})$. Let $\vec{\tau}_1,\vec{\tau}_2,\vec{\tau}_3,\ldots$ be an infinite sequence of strategies such that $\vec{\tau}_i$ is played by player 1 on the ith round of play, Q_0 be the identity matrix, and $(\forall i>0)(Q_i=Q(\vec{\sigma_1},\vec{\tau}_i))$. It should be

clear that the expected value of the game if these strategies are followed for up to n rounds is $\sum_{i=0}^{n}\prod_{j=0}^{i} Q_j\vec{v_0}$. Thus we can say:

$$\begin{aligned}
\sum_{i=0}^{n}\prod_{j=0}^{i} Q_j\vec{v_0} &\leq \sum_{i=0}^{n}\prod_{j=0}^{i} Q_j(\vec{v^*} - Q_{i+1}(\vec{v^\delta} \dot{-} \vec{\delta})) \\
&\leq \sum_{i=0}^{n}\prod_{j=0}^{i} Q_j\vec{v^*} - \sum_{i=0}^{n}\prod_{j=0}^{i} Q_jQ_{i+1}(\vec{v^\delta} \dot{-} \vec{\delta}) \\
&\leq \sum_{i=0}^{n}\prod_{j=0}^{i} Q_j\vec{v^*} - \sum_{i=1}^{n+1}\prod_{j=0}^{i} Q_j(\vec{v^\delta} \dot{-} \vec{\delta}) \\
&\leq \vec{v^*} + \sum_{i=1}^{n}\prod_{j=0}^{i} Q_j\vec{v^*} - \sum_{i=1}^{n+1}\prod_{j=0}^{i} Q_j(\vec{v^\delta} \dot{-} \vec{\delta}) \qquad (4.2.7) \\
&\leq \vec{v^*} + \sum_{i=1}^{n}\prod_{j=0}^{i} Q_j\vec{v^*} - \sum_{i=1}^{n}\prod_{j=0}^{i} Q_j(\vec{v^\delta} \dot{-} \vec{\delta}) - \prod_{j=0}^{n+1} Q_j(\vec{v^\delta} \dot{-} \vec{\delta}) \\
&\leq \vec{v^*} + \sum_{i=1}^{n}\prod_{j=0}^{i} Q_j(\vec{v^*} - (\vec{v^\delta} \dot{-} \vec{\delta})) - \prod_{j=0}^{n+1} Q_j(\vec{v^\delta} \dot{-} \vec{\delta}) \\
&\leq \vec{v^*} + \sum_{i=1}^{n}\prod_{j=0}^{i} Q_j(\vec{v^*} - \vec{v^\delta} + \vec{\delta}) - \prod_{j=0}^{n+1} Q_j(\vec{v^\delta} \dot{-} \vec{\delta})
\end{aligned}$$

From (4.2.7) we can see that, for a given ϵ and a δ such that $\delta \leq \frac{\epsilon}{2n}$, the expression $\vec{v^*} - \vec{v^\delta} + \vec{\delta} \leq \frac{\epsilon}{n} \cdot \vec{1}$. Again, because the rows of the Q matrices are probability distributions, the right-hand sides of the inequalities in (4.2.7) are bounded by $\vec{v^*} + \vec{\epsilon}$. This means that σ_2 is an ϵ strategy for player 2. □

Lemma 2.8 still doesn't give us quite what we want - determining the strategies for both players requires that we know the value of $\vec{v^\delta}$. Like $\vec{v^*}$, we cannot in general determine $\vec{v^\delta}$. However, we can do better!

Lemma 2.9. For any ϵ, given some $n \in \mathbb{N}$ which is an upper bound on the length (in rounds) of the game, there exists $\delta > 0$ and $k \in \mathbb{N}$ such that the vector $\vec{\sigma_2} \ni (\forall \gamma \in \Gamma \setminus \Gamma^T)[\vec{\sigma_2}[\gamma] = \sigma_2(\mathcal{N}(G(\gamma, \vec{v_k} \dot{-} \vec{\delta}))]$ is an ϵ strategy for Player 2.

Proof. Given $\epsilon > 0$ and $n \in \mathbb{N}$, by Lemma 2.6 there is a $\delta > 0$ such that $\vec{v^\delta} \geq \vec{v^*} - \frac{1}{3n} \cdot \vec{\epsilon}$. Now choose k such that $F_\delta^{\circ k}(\vec{v_0}) \geq \vec{v^\delta} - \frac{1}{3n} \cdot \vec{\epsilon}$. Using the

same logic from Lemma 2.8, we can say that $\vec{v_0} \leq \vec{v^*} - Q(\vec{\tau}, \vec{\sigma_2})(\vec{v_k} \dot{-} \vec{\delta})$ when $\vec{\tau}$ is an arbitrary strategy played by player 1. Let $\vec{\tau}_1, \vec{\tau}_2, \vec{\tau}_3, \ldots$ be an infinite sequence of strategies such that $\vec{\tau}_i$ is played by player 1 on the ith round of play, Q_0 be the identity matrix, and $(\forall i > 0)(Q_i = Q(\vec{\sigma_1}, \vec{\tau_i}))$. It should be clear that the expected value of the game if these strategies are followed for up to n rounds is $\sum_{i=0}^{n} \prod_{j=0}^{i} Q_j \vec{v_0}$. Thus we can say:

$$\begin{aligned}
\sum_{i=0}^{n} \prod_{j=0}^{i} Q_j \vec{v_0} &\leq \sum_{i=0}^{n} \prod_{j=0}^{i} Q_j (\vec{v^*} - Q_{i+1}(\vec{v_k} \dot{-} \vec{\delta})) \\
&\leq \sum_{i=0}^{n} \prod_{j=0}^{i} Q_j \vec{v^*} - \sum_{i=0}^{n} \prod_{j=0}^{i} Q_j Q_{i+1}(\vec{v_k} \dot{-} \vec{\delta}) \\
&\leq \sum_{i=0}^{n} \prod_{j=0}^{i} Q_j \vec{v^*} - \sum_{i=1}^{n+1} \prod_{j=0}^{i} Q_j (\vec{v_k} \dot{-} \vec{\delta}) \\
&\leq \vec{v^*} + \sum_{i=1}^{n} \prod_{j=0}^{i} Q_j \vec{v^*} - \sum_{i=1}^{n+1} \prod_{j=0}^{i} Q_j (\vec{v_k} \dot{-} \vec{\delta}) \\
&\leq \vec{v^*} + \sum_{i=1}^{n} \prod_{j=0}^{i} Q_j \vec{v^*} - \sum_{i=1}^{n} \prod_{j=0}^{i} Q_j (\vec{v_k} \dot{-} \vec{\delta}) - \prod_{j=0}^{n+1} Q_j (\vec{v_k} \dot{-} \vec{\delta}) \\
&\leq \vec{v^*} + \sum_{i=1}^{n} \prod_{j=0}^{i} Q_j (\vec{v^*} - (\vec{v_k} \dot{-} \vec{\delta})) - \prod_{j=0}^{n+1} Q_j (\vec{v_k} \dot{-} \vec{\delta}) \\
&\leq \vec{v^*} + \sum_{i=1}^{n} \prod_{j=0}^{i} Q_j (\vec{v^*} - \vec{v_k} + \vec{\delta}) - \prod_{j=0}^{n+1} Q_j (\vec{v_k} \dot{-} \vec{\delta}) \\
&\leq \vec{v^*} + \sum_{i=1}^{n} \prod_{j=0}^{i} Q_j ((\vec{v^*} - \vec{v^\delta}) + (\vec{v^\delta} - \vec{v_k}) + \vec{\delta}) - \prod_{j=0}^{n+1} Q_j (\vec{v_k} \dot{-} \vec{\delta})
\end{aligned} \tag{4.2.8}$$

Similarly to Lemma 2.8, the terms $(\vec{v^*} - \vec{v^\delta})$, $(\vec{v^\delta} - \vec{v_k})$, and δ are each less than or equal to $\frac{\epsilon}{3n} \cdot \vec{1}$ and the rows of the Q matrices are probability distributions. Therefore, the right-hand sides of the inequalities in (4.2.8) are bounded by $\vec{v^*} + \vec{\epsilon}$. □

N.B. that Corollary 2.7.1 and Lemma 2.8 suffice to prove the existence of ϵ-strategies, and that Corollary 2.7.2 and Lemma 2.9 together prove the theorem. □

There are several things worth noting here:

- When choosing a δ, making it less than $\frac{\epsilon}{3n}$ is a necessary but not sufficient condition for obtaining ϵ strategies. Also one cannot, in

general, know when they've reached an iteration k such that $\vec{v^{\delta}} - \vec{v_k} \leq \frac{\vec{\epsilon}}{3n}$ for any given δ. In practice, the best heuristic is to choose the smallest δ which won't exacerbate numerical errors and to iterate as much as time will afford. The above proof shows that doing so will lead to sane strategies for both players, even if the ϵ for such strategies cannot be known.

- For any game with an infinite state space, one must choose a finite subset of the state space to solve using Algorithm 4.1.
- Player 2's strategies hinge upon the game eventually ending (hence the requirement for the upper bound on the rounds). In practice, all games end so choosing an upper bound such that no game should reasonably expect to last that long is sufficient.

Chapter 5

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. In other words, the player will lose no matter what move (s)he makes, so there is no reason to choose one move over another. Similarly, as we shall see shortly, it is possible for the opponent to be able to win no matter what move (s)he chooses - so again, there is no reason to choose one move over another. If it doesn't matter what the players do, the outcome of the game is the same - the game loses its entertainment value. 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 an uninteresting 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 5.1, there are only certain positions from which Thok is guaranteed to capture Lil' Cal. If he's not in one of 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

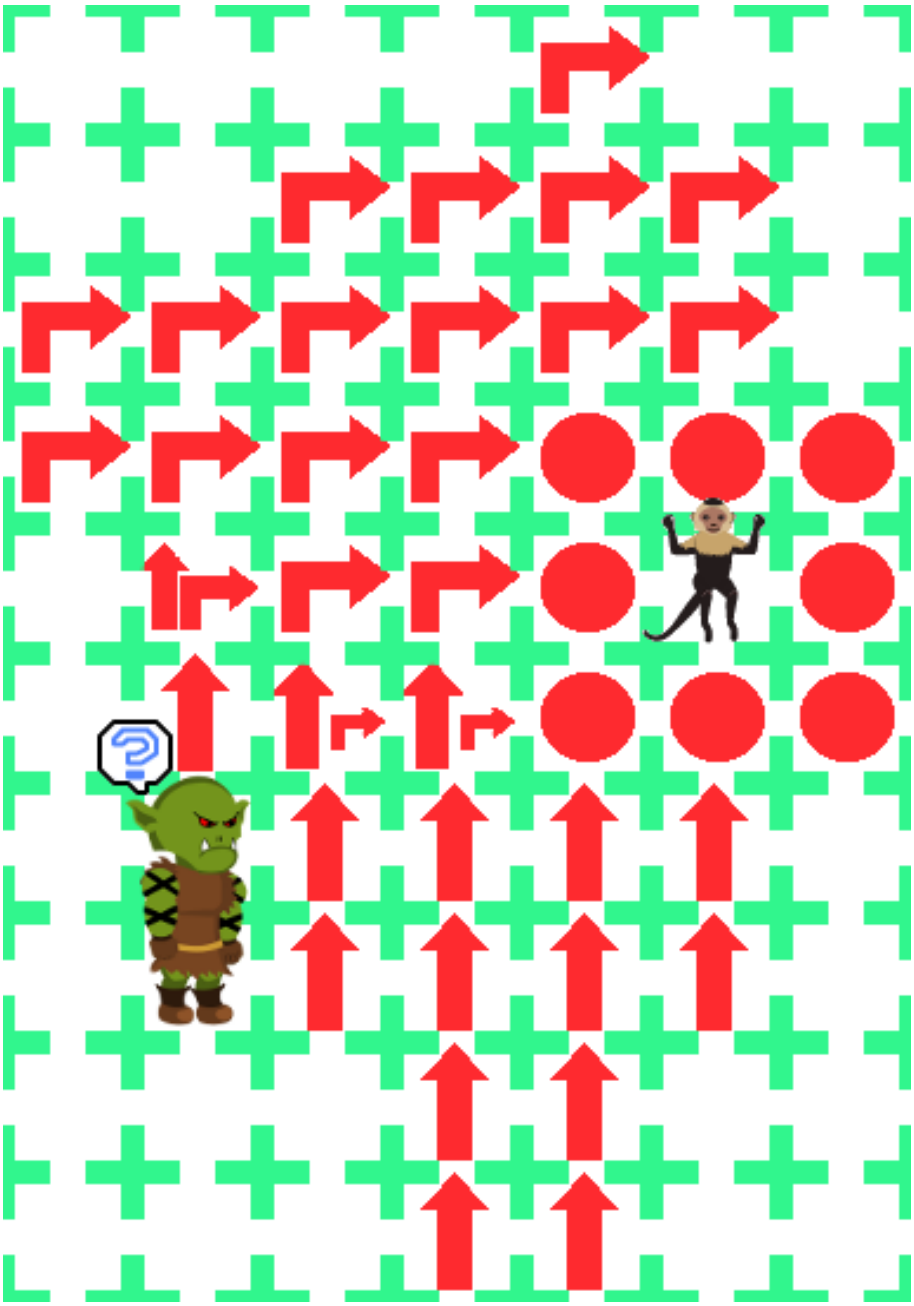


Figure 5.1: Policy for Thok shown graphically. The red arrows indicate what Thok should do when he finds himself in those squares. In the square containing the two arrows of equal sizes, Thok should choose roughly evenly between both options. In the squares with the large forward and small right arrow, Thok should employ an ϵ strategy - choosing forward with probability arbitrarily close to 1. Squares without arrows indicate that nothing Thok does will lead to capture if Lil' Cal plays his cards right.

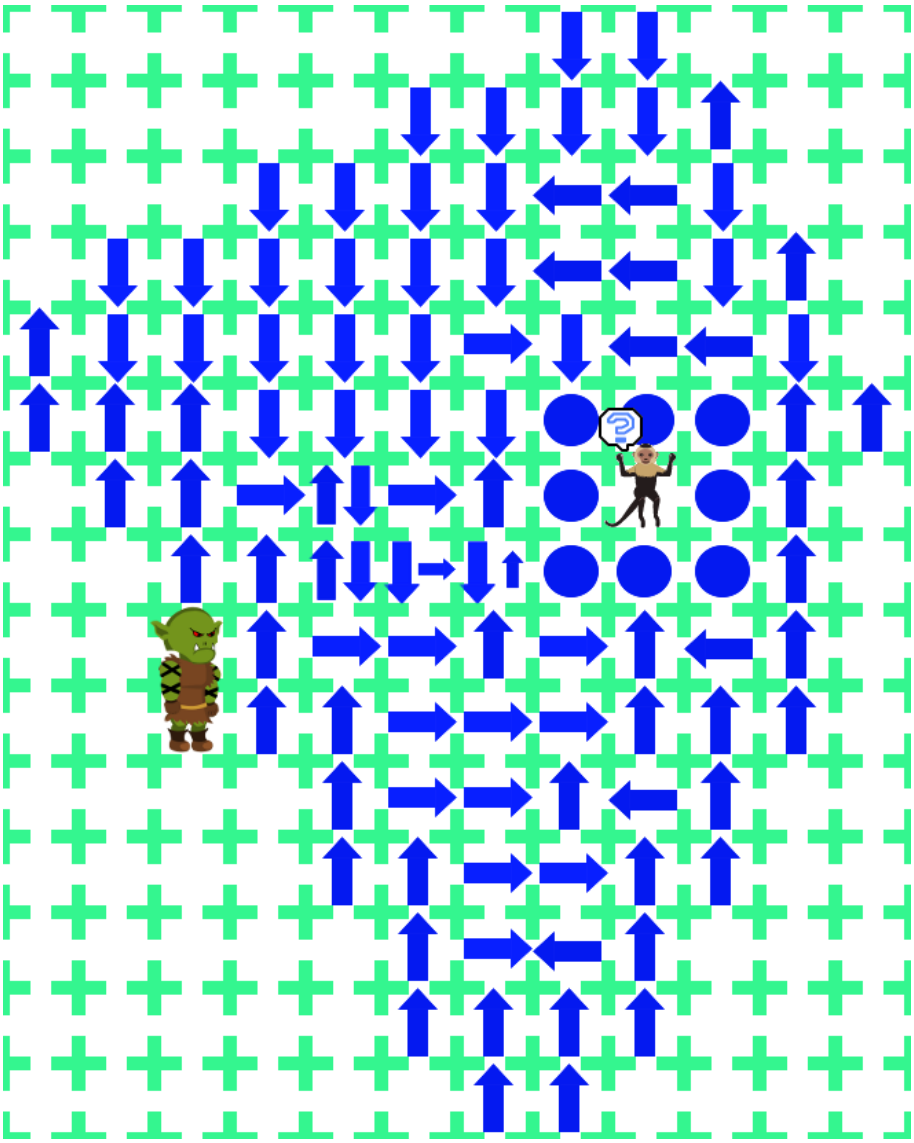


Figure 5.2: Policy for Lil' Cal shown graphically. The blue arrows indicate what Cal should do when Thok is in those squares. In the squares containing the two arrows of equal sizes, Cal should choose roughly evenly between both options. In the squares with the large down arrow and small right/up arrow, Cal should employ an ϵ strategy - choosing to move down with probability arbitrarily close to 1. Squares without arrows indicate that Cal can do whatever he wants when Thok is there and still evade capture.

as possible regardless of whether or not he's guaranteed to catch the monkey. Using just the definition of a Nash equilibrium and Algorithm 4.1[1], 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.

Intuitively, then, a game is most interesting when both players have to choose their moves carefully to achieve a desirable outcome. In the case of our games of survival, this means that the interesting states of the game are those for which the game value is greater than zero. It is these states that require player 1 to play a Nash equilibrium policy in order to achieve that game value, and for player 2 to play a Nash equilibrium policy to win the game (if the game value from player 1's point of view is less than 1) or to keep the game going as long as possible. As we saw with Lil' Cal, there may be states where player 2 must still play a particular policy even though the game value for that state is zero. But there may also be game states where it doesn't matter what policy player 2 follows; no matter what moves the players choose, player 2 can still force a game value of zero from the next state. Thus, the further a game state is from the ones that have positive game values, the less interesting it is.

5.1 Cooperative Reachability

Thus we need to characterize these less interesting game states in terms of how far they are from the more interesting states. If both players want to get from a less interesting state to a more interesting state, they

[1] For all results obtained from running Algorithm 4.1 in this chapter, the values $\delta = 0.001$ and $k = 1000$ were used.

will have to choose moves in a cooperative manner. Hence we define the concept of *Cooperative Reachability*.

- The *attractor set* $R_0 \subset \Gamma$ is the set of states deemed most interesting.
- set $R_n = \{\gamma \in \Gamma \mid (\exists c_1 \in C_1(\gamma))(\exists c_2 \in C_2(\gamma))[f(\gamma, c_1, c_2) \in R_{n-1}]\} \cup R_{n-1}$

Thus the desired states can be *cooperatively reached* from a state in R_n in n rounds of cooperative play.

How should the players decide on a policy when the game is in a state γ in R_n? To answer this question, we propose that a "cooperative" non-zero sum game $\mathcal{G}_C$ be analyzed. We define $\mathcal{G}_C$ as follows:

Let $n \geq 1$ be the unique positive number such that $\gamma \in R_n \setminus R_{n-1}$. Then $\mathcal{G}_C = (N, (C_1(\gamma), C_2(\gamma)), h)$ where

$$h(x,y) = \begin{cases} 1 & \text{if } f(\gamma, x, y) \in R_{n-1} \\ 0 & \text{otherwise} \end{cases} \tag{5.1.1}$$

Note that in this game, both players' payoff is h, rather than the usual player 2's payoff being $-h$. Since this is a non-zero sum game, finding some Nash equilibrium for the game is not good enough. Because both players receive the same payoff in any situation, they must both choose policies that maximize h. We propose that this be done as follows:

Let $r(h,x) \stackrel{\text{def}}{=} \{y \mid h(x,y) = 1\}$ and likewise $c(h,y) \stackrel{\text{def}}{=} \{x \mid h(x,y) = 1\}$. Let $\texttt{rmin} \stackrel{\text{def}}{=} \min_x |r(h,x)|$, $\texttt{rmax} \stackrel{\text{def}}{=} \max_x |r(h,x)|$, $\texttt{cmin} \stackrel{\text{def}}{=} \min_y |c(h,y)|$, and $\texttt{cmax} \stackrel{\text{def}}{=} \max_y |c(h,y)|$. If $\texttt{rmax} > \texttt{rmin}$, let D_1 be the set $\{x \mid |r(h,x)| > \texttt{rmin}\}$, else D_1 = the domain of the first argument of h. If $\texttt{cmax} > \texttt{cmin}$, let D_2 be the set $\{y \mid |c(h,y)| > \texttt{cmin}\}$, else D_2 = the domain of the second argument of h. Define function $h' : D_1 \times D_2 \mapsto \{0,1\}$ such that for $x \in D_1$ and $y \in D_2$, $h'(x,y) = h(x,y)$. This construction of a new utility function is repeated on h' and so on until $\texttt{rmax} = \texttt{rmin}$ and $\texttt{cmax} = \texttt{cmin}$. In other words, if h is represented by a matrix, the rows with the fewest 1s and the columns

with the fewest 1s are dropped simultaneously, until no more such rows or columns can be dropped without making the matrix vanish completely. The players should then choose randomly from the moves represented by the remaining rows and columns.

5.2 Finding Policies

Using the above definitions, we will now describe several ways to find better AI policies for *both* players when in a situation where the game is unwinnable for one of the players. We will use Right Turn to illustrate these methods, however it should be clear to the reader that they are generally applicable in games where such situations arise.

To find a policy for Lil' Cal:

1. Solve the game using Algorithm 4.1, save the policy table obtained for the monkey.

2. Let $\vec{v}$ be the vector obtained in Step 1. With attractor set $\{\gamma \mid \vec{v}[\gamma] \neq 0\}$, for each $n > 0$ and $\gamma \in R_n \setminus R_{n-1}$, solve the cooperative game $\mathcal{G}_C$ and save the policy table for these states for the monkey.

These policy tables can be combined to produce behavior for Cal which will lead to more interesting game play. For states in R_0, Cal should play the policy given by Algorithm 4.1. For states in $R_1 \setminus R_0$, Cal should stochastically choose between the policy given by Algorithm 4.1 and that given by Step 2.[2] For all other states, Cal should play the policy given by Step 2.

Following the above procedure will give the monkey the behavior of moving *toward* the orc when they are far apart, prolonging the game as

[2] The higher the frequency with which the monkey chooses the policy for the original game, the more difficult it becomes for the orc to capture the monkey.

long as possible when inside the orc's winnable region, and sometimes making a mistake at the border of the winnable region to allow the orc a chance of victory. Thus, the set of interesting states (those with a positive game value for player 1) has been increased to virtually the whole set of states in the game. (Thok and Cal will move toward each other until a state in R_1 is reached, where the new game value is the same as the rate at which Cal chooses to make a mistake.)

To find a policy for Thok:

1. Solve the game using Algorithm 4.1, save the policy table obtained for the orc.

2. Let $\vec{v}$ be the vector obtained in Step 1. With attractor set $\{\gamma \mid \vec{v}[\gamma] \neq 0\}$, for each $n > 0$ and $\gamma \in R_n \setminus R_{n-1}$, solve the cooperative game $\mathcal{G}_C$ and save the policy table for these states for the orc.

Combining these tables to produce a strategy for the orc is straightforward: For states in R_0, the orc should play the policy yielded in Step 1. For all other states, he should play the policy given by Step 2.

The above procedure will give the orc the behavior of always moving *toward* the monkey, regardless of whether or not he has a chance of victory. Clearly, the chance that the game winds up in the critical region is higher than if the orc just does the random walk given in the original policy.

5.3 Discussion of Results

We actually ran this on a few games. Namely Right Turn, 1D Hero vs. Bear, and 2D Heroes vs. Bears (with one hero and one bear, on a 7×7 grid, with the powder at location $(4, 4)$). We got some interesting results, and we shall explain them somewhat here!

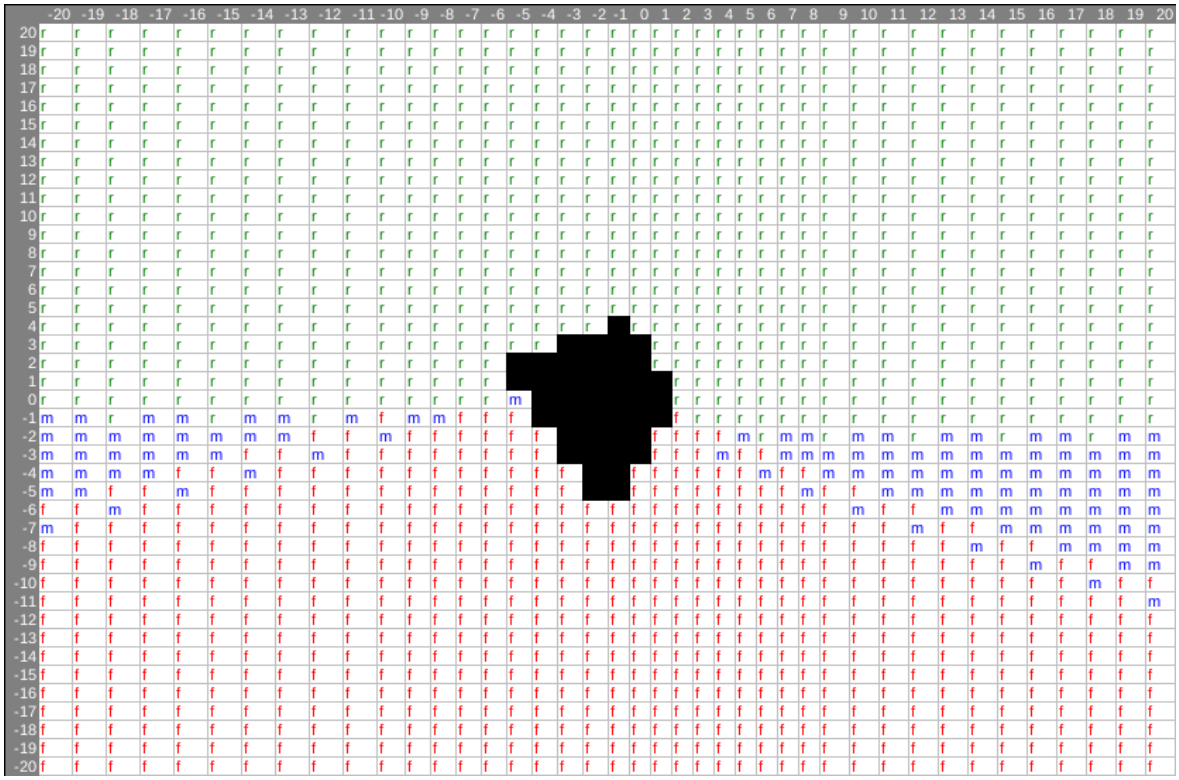


Figure 5.3: Thok's policy according to Game Theory of Lowered Expectations. The squares with m denote ones in which Thok should choose between both of his options with equal probability. In most of these squares, the reality is that he can select either action and still end up closer to the attractive region.

5.3.1 Right Turn

Figure 5.3 shows Thok's policy in a 41×41 region surrounding R_0. The triangular sections of mixed strategies expanding out to the left and right of R_0 denote states in which, assuming cooperation from the monkey, the orc will get closer to R_0 no matter what he does. We can simplify the policy by changing the action in these states to be what Thok should choose in the surrounding states, as in Figure 5.4. With the exception of

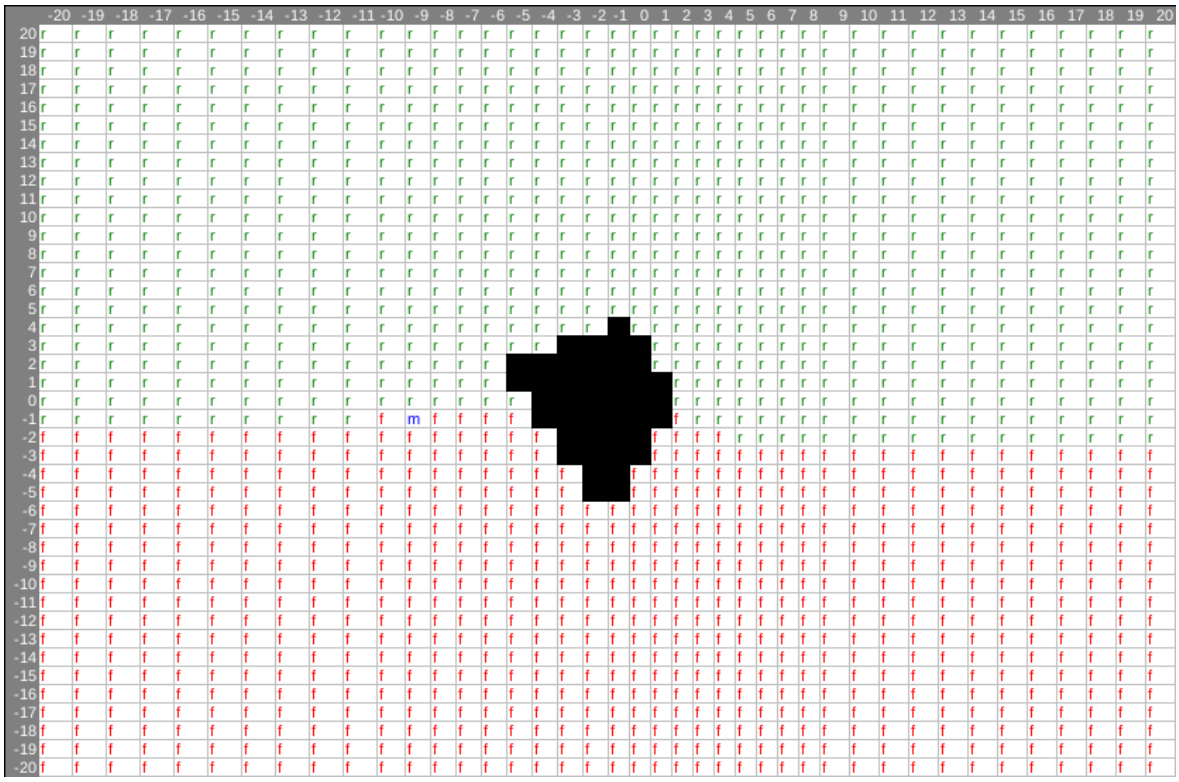


Figure 5.4: Thok's *normalized* lowered expectations policy. Note that there is still a single state in which he should employ a mixed strategy. This policy can be generalized/described more simply by Algorithm 5.1.

Algorithm 5.1 Thok's lowered expectations strategy described algorithmically

```
function CHOOSE_ACTION(γ)
    if γ = (−9, −1) then
        return RANDOM_CHOICE((Forward, Right))
    else if y(γ) ∈ ℕ then
        return Right
    else if x(γ) ∉ [−10, 2] ∧ y(γ) = −1 then
        return Right
    else if x(γ) > 4 ∧ y(γ) = −2 then
        return Right
    else
        return Forward
    end if
end function
```

$(-9, -1)$[3], Thok can employ a pure strategy in every state outside of R_0. Algorithm 5.1 describes Thok's policy in code. Interesting to note is that this policy follows roughly the intuitive idea of "if the monkey is in front of the orc, he should move forward, otherwise he should turn right."

Figure 5.5 shows Cal's policy after simplifying states where it doesn't matter what the monkey does. Of note is that the simplification process for Cal's policy was a bit more complicated than for Thok's, however the process is still algorithmic. For each state in which Cal has more than one choice of action:

1. Remove all actions which do not take Cal closer to the current location of the orc.

2. Choose the first remaining action in order of precedence: *Down*, *Left*, *Right*, *Up*. What is left is a policy which has Cal move toward the orc when he's far away.

[3] This is because the reduced matrix for $(-9, -1)$ still contains 0s. In such situations, the "correct" choice for player 1 hinges upon player 2's choice and vice-versa. Thus, a mixed strategy must still be employed.

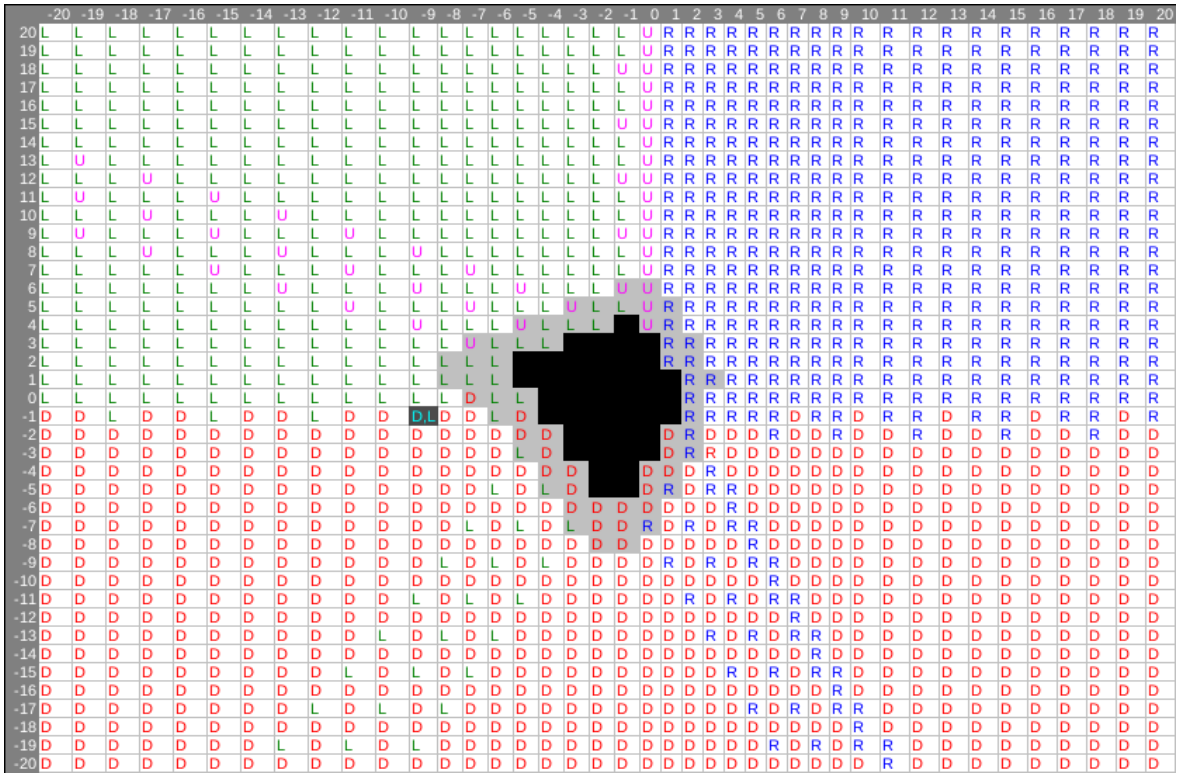


Figure 5.5: Lil Cal's *normalized* policy according to Game Theory of Lowered Expectations. The squares with light gray background surrounding R_0 correspond to states in R_1. In these states, Cal will be giving up the game if he chooses the action suggested by this policy.

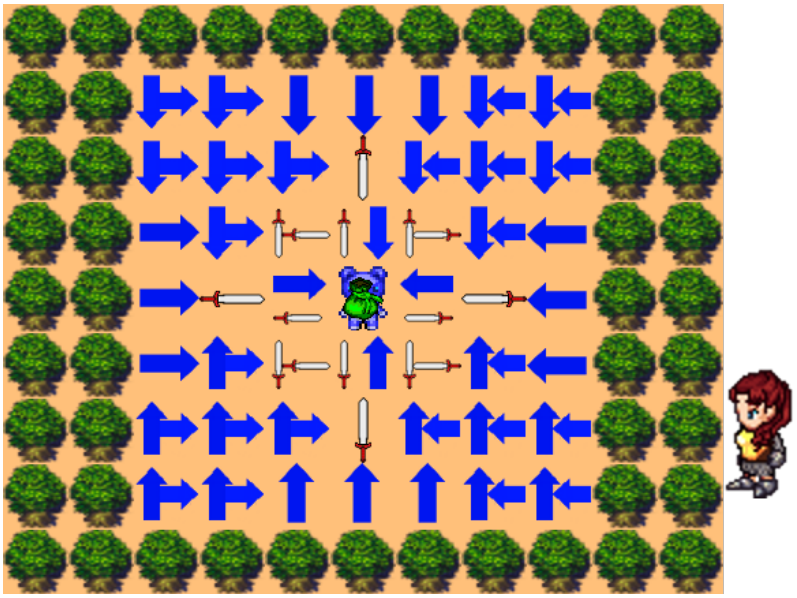


Figure 5.6: Eve's policy on a 7×7 grid, with a bear and the powder at location $(4, 4)$, when Eve and the bear have the same amount of HP remaining.

The states with a light gray background are those which are one step away from R_0 - these are states in which Cal will be throwing the game if he follows the policy in Figure 5.5. An AI opponent playing the role of Cal should stochastically choose between the policies in Figures 5.5 and 5.2. The higher the probability with which the AI picks lowered expectations policy, the easier it becomes for the orc to catch the monkey.

5.3.2 Bear Brawl

Figures 5.6 and 5.7 are a sampling of Eve's lowered expectations policy on a 7×7 grid with the powder located in the center. This policy was generated according to the procedure laid out in Section 5.1, using the states in which Eve has a winning policy according to Algorithm 4.1 (with $\delta = 0.001$ and $k = 1000$) as R_0. In both figures, a single bear remains and guards the powder.

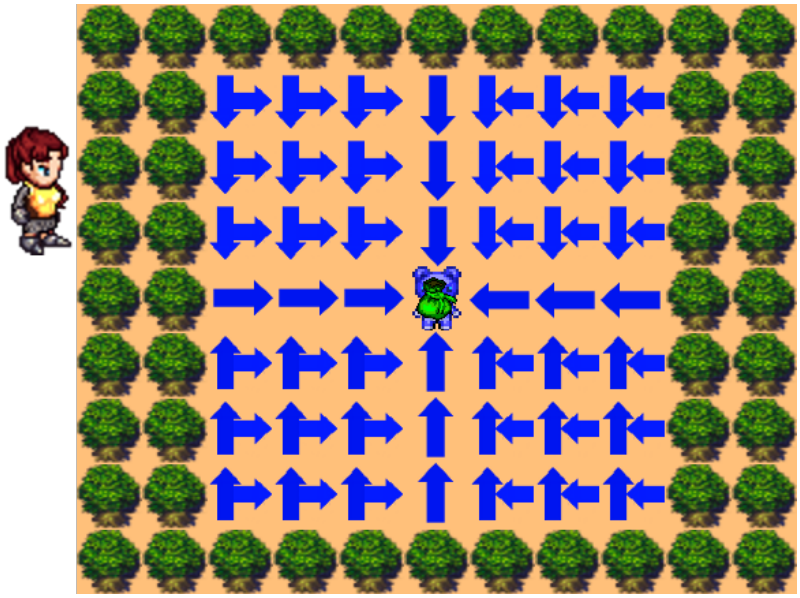


Figure 5.7: Eve's policy on a 7×7 grid, with a bear and the powder at location $(4, 4)$, when the bear has more HP remaining than Eve.

Figure 5.6 is the policy Eve should follow when she and the bear have the same HP value ($z(\text{Eve}) = z(\text{bear}) = n$ for some $n \in \mathbb{N}$). Of note are the squares with a Manhattan distance of 2 from the powder. On these squares, Eve should stand her ground and attack in the direction of the bear. Should the bear charge toward her, he'll be running into Eve's blade. This will give Eve the advantage (or even the win, when $n = 1$). Should the bear move in some other direction, Eve's policy for the resulting state will have her move closer to the powder.

Figure 5.7 is Eve's policy if the bear has more HP than she ($z(\text{Eve}) < z(\text{bear})$). Here, the policy can be summarized as "move toward the bear/powder".

We also generated policies for the 1 dimensional case. For both the hero and the bear, the policy is quite simple - "If at the powder, attack. Otherwise, move toward the powder". This matches intuition.

5.3.3 Other Observations

In some situations, there is more than one option for one or both players that will result in the game getting closer to the attractive region. In some cases, the reduced $\mathcal{G}_C$ matrix contains only 1s. When this happens, it is probably best to chose a pure strategy that fits into the pattern of pure strategies already admitted for other states. An example of this is the procedure we followed to generate the policy for Lil' Cal showcased in Figure 5.5. In the cases where the reduced $\mathcal{G}_C$ matrix still contains some 0s, the players must still choose randomly from among the options. If they choose pure strategies, they might choose moves that give one of these 0s, and they will not get closer to the attractor set.

Chapter 6

CONCLUSION AND RELATED WORK

Computer games with simultaneous or real-time action are an interesting topic of study in the fields of Game Theory and Artificial Intelligence. In fact, these games exist at the confluence of these two fields. This dissertation has explored what the existing literature has to say about solving deterministic simultaneous action games with perfect information, as well as offer several new contributions to this literature.

The rest of this chapter is divided into several sections: In Section 6.1 we briefly explore related work in the literature and outline the differences between this work and ours, in Section 6.2 we discuss several ways in which the work in this dissertation could be built upon in the future, and in Section 6.3 we summarize the contributions made by this dissertation and offer some concluding remarks.

6.1 Related Work

Despite the efforts of researchers such as Sue Epstein [14], 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[48].

Work has been done in the Reinforcement Learning community; for example Gabor et al in [17] use an iterative learning process to develop strategies for multi-state games. 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. Nevertheless, reinforcement learning work is relevant because the iteration formulas they use resemble the iteration formulas discussed in this dissertation.

Traditional planning based on STRIPS [16] 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 work such as [42], only changes via forces 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 [12].

Daniel Andersson deals with turn-based recursive games [3], 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 [29].

Every year, AAAI hosts a General Game Playing competition [18]. 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 [22, 9, 41, 43, 30, 25, 23, 5], use of recursive game theory appears to be completely absent.

Though some of the terminology used in our game theory of lowered expectations is inspired by that of reachability and safety games, the focuses of these two theories are quite different. Reachability and safety games are infinite games (in that they are expected never to terminate) in which a player aims to drive the game to a certain set of states and remain there (reachability games), or in which a player's goal is to perpetually avoid a particular set of states (safety games). These games are competitive and are generally used to model systems in control theory[2], but have also been applied to other domains[37]. They have also been described as special cases of recursive games[7, 10]. In terms of what we have done, games of survival are reachability games from the point of view of player 1, and safety games from the point of view of player 2. In game theory of lowered expectations, the attractor set can be looked at as the unsafe region of a safety game. As such, the goal for both players in game theory of lowered expectations is to get the game *out* of the safety region.

The concurrent reachability games dealt with by Chatterjee et al in [7] can be mapped directly into games of survival. The technique applied in that paper, however, can produce strategies that will not necessarily lead a player to a goal state in the games we're interested in. For a simple, illustrative example we consider a version of 1D1H1B where Eve must reach the powder to win regardless of the state of the bear. In this instance of the game, there are four locations: $0, 1, 2, 3$, with the powder at location 0. The critical vector for this game when the bear is dead is $\vec{v}^* = (1, 1, 1, 1)$, and is the only vector within ϵ of itself for any $\epsilon < 1$ in the sequence of vectors defined by iterating F from $\vec{v}_0 = (1, 0, 0, 0)$. Chaterjee et al rely upon finding a vector $F^{\circ i}(\vec{v}_0)$ within ϵ of the critical vector and using that to obtain strategies. For this particular game, there is no indication that Eve would be any better off moving left than moving right (*e.g.*, when at location

2). However, using Algorithm 4.1 with $\delta = 0.1$ and $k = 3$, the resulting vector $(1, 0.9, 0.8, 0.7)$ will produce the strategy that Eve should always move left.

6.2 Future Work

The work done in this dissertation opens several doors for new research opportunities. In this section, we briefly describe several topics that could be researched/worked on which would build upon or extend the work done in this dissertation. This list is by no means exhaustive - it merely contains those things to which we've given more than just perfunctory thought.

6.2.1 Producing a Program from a Policy Table

When we initially started the research that eventually turned into this dissertation, we had a goal of producing work useful to the computer gaming industry. In order for this theory to be useful in creating interesting and engaging AI opponents in real games, there need to exist software systems which make use of it.

One possible system 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 briefly describe a method for generating the policy table for a finite, tractable portion of a game's states to use as input to a classification algorithm. As an example, see Table 6.1 for a (small) sample policy table for 1D1H1B as formalized in Section 3.5.

The way in which this method defines the set of states to use in generating the policy table is inspired by the analysis of discrete differential games in [20]. We start from some subset of the terminal states (possibly all of them) and progressively work backward, adding states to the set

of states which are used as input to Algorithm 4.1 to generate the policy table.

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 tractable, 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, with techniques such as *decision tree induction* and *inductive logic programming*. Following is a list of algorithms which would be good candidates for use in an offline planner:

1. OC1 [33]
2. C4.5/C5.0 [38]
3. Progol [32]
4. FOIDL [31]

OC1 and C5.0 are two leading decision tree induction systems, and Progol and FOIDL are two leading inductive logic programming systems.

6.2.2 Using forward Search with a heuristic evaluation function

Another possible application is to allow the game agent to use Algorithm 4.1 to make a decision on the fly about what to do next. Online planning has been studied in the literature. Examples include [5, 1, 6, 42, 47]. The system described 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 would be generated using Algorithm 6.1, then solved using Algorithm 4.1. The function h referenced in Algorithm 6.1

Table 6.1: Example policy table for 1d1h1b

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

is some *heuristic evaluation function* which attempts to approximate the value of the game at the state which is its argument. This h would also be used to provide the values for the initial vector input to Algorithm 4.1. Plenty of work has been done in constructing heuristic evaluation functions for games [22, 9, 30, 25]. Ideally, h should be admissible (in the classical sense).

We conjecture that for two distinct admissible heuristics h_1 and h_2, the iterative process of Algorithm 4.1 would converge to the same vector of game values whether it starts with an initial vector determined by h_1 or by h_2, and the vector obtained after k iterations will be closer to that vector limit when starting from h_1 than h_2 when h_1 dominates h_2. So the vector obtained by starting from h_1 will be closer to the critical vector than the vector obtained by starting from h_2. This has to be tempered by the fact that the iterations are being done over a small set of the games states instead of all of them.

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. Furthermore, if a positive game value is found for the current game state from which the graph was generated, the policy obtained by Algorithm 4.1 is the policy that will be played. If the game value turns out to be zero, our methodology for game theory of lowered expectations will be applied to the states in the graph, and the policy so derived for the current state will be played.

Algorithm 6.1 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($\Re^\Gamma, \gamma$)
  $Q \leftarrow []$
  $T \leftarrow \{\}$
  $S \leftarrow \{\}$
  $Push(Q, \gamma)$

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

    for all $\alpha \in C_1(curr)$ **do**
      for all $\beta \in C_2(curr)$ **do**
        $succ \leftarrow f(curr, \alpha, \beta)$

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

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

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

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

6.2.3 Symbolic Game Theory

The systems proposed in both Sections 6.2.1 and 6.2.2 are both limited by the size of the game they're attempting to solve. The fact that computational resources are still very limited in today's world prevents the theory presented in this dissertation from being generally applicable in practice to the types of games that people play.

We wonder if a symbolic approach can be developed for solving games with large state spaces. For many of the games we analyzed through the course of this dissertation, we were able to (manually) divide the state space into a small number of partitions and determine a policy to be followed for each partition. The policies we came up with were very close to those output by Algorithm 4.1 and the game theory of lowered expectations. For example, when thinking about the one hero one bear games, it became obvious that for both players, whichever one is closer to the powder, that player's best strategy is to go towards the powder - and in the case of the bear, to defend it once there. When this guarantees a win for the bear, such as when both players have the same number of hit points, the bear should occasionally move away from the powder to give the hero a chance to win in accordance with our game theory of lowered expectations. We saw this just by looking at the rules of the game, and not by manually doing many simulations of game play or the iterations of Algorithm 4.1. What we'd like to determine is if it's possible to a) algorithmically divide a game's state space into a small number of partitions in general and b) apply game theory to these partitions using symbolic reasoning rather than numeric reasoning.

6.2.4 A Theory of Interesting Play

The ultimate goal of any AI controlled agent in a computer game is to make the game an engaging and enjoyable experience for the humans playing it. This is reflected in our choice of games of speed for modeling computer games. In a game of speed, the computer has as much incentive to simply get in the human's way of winning as it does to win the game itself.

However, that alone does not guarantee that play will be interesting. How does one formalize a subjective concept such as interesting? Sid Meier stated that a (good) game is "a series of interesting choices"[39]. Further, he defined three requirements a choice must satisfy to be considered interesting[21]:

1. No single option is clearly better than the other options.
2. The options are not equally attractive.
3. The player must be able to make an informed choice.

In terms of recursive games, we say that state γ is *very interesting* if all three of the above rules hold in γ. If rule 3 and one of rule 1 or 2 holds in γ, then γ is *mildly interesting*. We'd like to develop a theory for identifying which regions of the game are mildly interesting and which are very interesting, as well as strategies for keeping the game in or around states which are interesting.

To some extent our game theory of lowered expectations is already such a theory, but we would like to formalize the relationship between interestingness and the rules of a game so that game designers will have some guidance when crafting the rules of a new game. Such theory has been treated informally in work such as [24]; we would like to see a more formal treatment that would be useful to game designers.

6.3 Closing Remarks

The goal of this dissertation was to provide a theory for deterministic simultaneous action games with perfect information. We firmly believe that we have accomplished this goal, and as such will briefly reiterate the components that went into it.

In Chapter 2 we introduced the Bear Brawl class of games. These games have several of the properties exhibited by real games, yet are simple enough to use as examples for research into game AI. We also defined a simultaneous action variant of the already interesting Hamstrung Squad Car game. Throughout the dissertation, we drew attention to and examined several interesting properties of this variant.

In Chapter 4 we defined games of speed, and showed an equivalence between them and games of survival, the latter of which have been studied in the literature. Most importantly in Chapter 4 is the iterative algorithm for obtaining sound strategies for both players in a game of survival, along with the proof of its correctness. This original proof verifies once and for all that in practice it is possible to compute strategies for both players in a recursive game of survival, with no a priori knowledge of the value of the game.

In Chapter 5, we drew attention to situations in recursive games where one or both players are left with the uninteresting decision between strategies which are all equally bad (or good). We defined game theory of lowered expectations as a means for players in such positions to obtain strategies which give those players a better chance at capitalizing upon the mistakes of their opponents, to more quickly end the game, or bring it to an interesting state.

It is our sincere hope that this dissertation will be of interest to researchers in both the artificial intelligence and game theory communities;

that they will find it engaging and thought provoking, and that further research will be built upon it. We also hope that software engineers in the computer gaming industry will find the theory presented here as a useful starting point into designing better AIs which will increase the immersiveness and entertainment value of the games that they build.

BIBLIOGRAPHY

[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] Rajeev Alur, P. Madhusudan, and Wonhong Nam, *Symbolic computational techniques for solving games*, STTT **7** (2005), no. 2, 118–128.

[3] Daniel Andersson, Kristoffer A. Hansen, Peter B. Miltersen, and Troels B. Sørensen, *Simple Recursive Games*, CoRR **abs/0711.1055** (2007).

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

[5] 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.

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

[7] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger, *Strategy improvement for concurrent reachability games*, Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11-14 September 2006, Riverside, California, USA, 2006, pp. 291–300.

[8] 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.

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

[10] Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman, *Concurrent reachability games*, Theoretical Computer Science **386** (2007), no. 3, 188–217.

[11] V. K. Domanskii, *Game of survival*, URL: http://www.encyclopediaofmath.org/index.php?title=Game_of_survival&oldid=33815, October 2014, Encyclopedia of Mathematics.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

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

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

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

[21] Jesper Juul, *Half-Real: Video Games Between Real Rules and Fictional Worlds*, The MIT Press, 2005.

[22] 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, pp. 93:1–93:7.

[23] Tomoyuki Kaneko, Kazunori Yamaguchi, and Satoru Kawai, *Abstract Automatic Feature Construction and Optimization for General Game Player*, Game Programming Workshop, no. 14, October 2001, pp. 25–32.

[24] Raph Koster, *A Theory of Fun for Game Design*, O'Reilly Media, December 2013.

[25] 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, July 2006, pp. 1457–1462.

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

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

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

[29] Peter B. Miltersen, *A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament*, Autonomous Agents & Multiagent Systems/Agent Theories, Architectures, and Languages, 2007, pp. 191–198.

[30] 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.

[31] 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.

[32] Stephen Muggleton, *Inverse entailment and Progol*, New Generation Computing **13** (1995), no. 3-4, 245–286.

[33] 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.

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

[35] John Nash, *Non-Cooperative Games*, The Annals of Mathematics **54** (1951), no. 2, 286–295.

[36] Michael Orkin, *Recursive Matrix Games*, Journal of Applied Probability **9** (1972), no. 4, 813–820.

[37] Dominique Perrin and Jean-Éric Pin, *Infinite words : automata, semigroups, logic and games*, Pure and applied mathematics, Academic, London, San Diego (Calif.), 2004.

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

[39] Andrew Rollings and Dave Morris, *Game Architecture and Design: A New Edition*, New Riders Games, 2003.

[40] Jonathan Schaeffer, Neil Burch, Yngvi Bjornsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Paul Lu, and Steve Sutphen, *Checkers Is Solved*, Science **317** (2007), no. 5844, 1518–1522.

[41] 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.

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

[43] 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.

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

[45] Frank Thuijsman and O. J. Vrieze, *Note on recursive games*, Game Theory and Economic Applications (1992), 133–145.

[46] 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.

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

[48] David E. Wilkins, *Using Patterns and Plans in Chess*, Artif. Intell. **14** (1980), 165–203.

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