Modeling an Adversarial Poacher-Ranger Hybrid Game

Maia Iyer
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
maiai@andrew.cmu.edu

Benjamin Gilby
School of Computing and Information
University of Pittsburgh
Pittsburgh, PA 15213
beg59@pitt.edu

Abstract

Pursuit-evasion differential games have been of special interest in applications of robotics and security in continuous space. A special class of pursuit evasion games is Active Target Defense Differential Games (ATDDGs), which incorporate a third entity into the game. This work considers, for the first time in the literature to the authors' knowledge, work on Partially-Observable Active Target Defense Differential Games (POATDDGs), which incorporate imperfect *state observability* and *structural observability*. We demonstrate reasoning about these formulations as hybrid games in the KeYmaeraX framework to prove winning strategies. This work presents basic examples of games and matching winning strategies, using few assumptions. We then show how to extend the basic game with additional assumptions on ranger mobility and prey cooperation to demonstrate how to apply the basic game to more specific domains and use-cases. We finally discuss the challenges of game proofs as strategy complexity increases, and we present additional extensions that may prove to be interesting future research questions such as extending the game to have more targets or pursuers or asymmetric terrain.

1 Introduction

Hybrid games [1][2] combine discrete, continuous, and adversarial dynamics in an environment. Many real-world applications involving continuous space and discrete control can be framed in the form of a hybrid game. This work frames the adversarial dynamic of park rangers, poachers, and prey in the context of hybrid games, where the goal of rangers is to protect prey from being caught by poachers. This is a natural choice because of the continuous component of elements in space, the discrete component of strategy and decisions for movement, and the adversarial component of rangers against poachers. This problem falls under the class of pursuit-evasion games, and more specifically, active target defense differential games, in which there is a third-party defender involved [3] [4].

This game is primarily difficult in real-world applications because often the ranger has a known amount of limited resources and an attacking poacher has an unknown, potentially unlimited amount of resources. Prior work has attempted to use game theory with stochastic strategies to decrease the win-rate of poachers—elements of randomness can make decisions more difficult for an adversary who greatly prefers to not get caught [5]. However, randomized strategies are generally difficult to implement in the real world. There are other works involving looking for winning pure strategies in the pursuit-evasion game assuming full-observability of entities in space, as well as full knowledge of structure and hyperparameters. The drawback to these works is they often require many assumptions on the exact structure, and also do not account for the asymmetry that naturally occurs in real-world applications.

In this work, we use previously built frameworks for reasoning about hybrid games. With certain modeling decisions within the KeYmaeraX framework [1], we are able to naturally consider an asymmetric game in terms of information. This allows us to extend the game of active target defense to an imperfect or delayed information game. We refer to this class of games as Partially-Observable Active Target Defense Differential Games (POATDDGs). Additionally, using this framework also allows us to control the amount of cooperation possible between target and defender, which has not generally been done in the context of active target-defense games.

The remainder of this paper is structured as follows: Section §2 describes several relevant previous works involving domain knowledge on poaching mitigation techniques, game analysis, and the specific pursuit-evasion games. Section §3 discusses the rules of the game. Section §4 discusses the current formal model of the game and desired properties for the system. Section §5 details the current modeling and proof progress done, linking to a Github with relevant .kyx files. Finally section §7 details extensions for the model we consider and would like to prove properties about.

2 Related Works

This section discusses relevant related works involving domain knowledge, hybrid games, and the specific class of pursuit-evasion games and active target defense games, as well as contributions of our work.

2.1 Poach Mitigation in the Real World

Much of the inspiration for this project derives from the real world problem of anti-poaching methods. In this domain, there have been several studies for modeling environmental factors affecting the likelihood of poaching including elevation, water, roads, tree cover, and distance to ranger camp [6]. Challenges of limited resources, visibility, mobility, and lack of knowledge about poacher location and resource make the task of protecting prey difficult.

A classical game theoretic approach to this problem often would involve stochastic strategies to decrease the chance of attacking adversaries to successfully obtain a target [5]. This work, however, uses hybrid game formulation to augment the fidelity of motion dynamics and to simplify the problem and identify strategies and assumptions necessary to guarantee success of ranger efforts. We believe this extends to other domains of interest in the differential game realm as discussed in §2.3 and §2.4.

2.2 Hybrid Games

Differential games relate to the analysis of games involving a state that evolves over time. These were first discussed and formalized by Rufus Isaacs [3] which has introduced related problems such as the homicidal chauffeur game, the princess and the dragon, and the lady in the lake.

Hybrid games are a class of problems that combine discrete, continuous, and adversarial dynamics [1]. Often, these games are formulated as being two-player and zero-sum, where the question is whether some player has a winning strategy. These are both determined and consistent, i.e., at every state, at least one player has a winning strategy. In previous work, dynamic game logic was introduced as a sound and complete axiomatization for reasoning about hybrid games [1].

In this work, we leverage dynamic game logic to reason about our formulated ranger-poacher-target game.

2.3 Pursuit-Evasion Games

A substantial amount of work revolving around differential games naturally falls under the class of pursuit-evasion games. Canonically in these games, there are two players—one is a pursuer, and one is an evader. The objective of pursuer is to catch the evader, whereas the objective of the evader is to avoid the pursuer. This particular dynamic of game has generated much interest in applications of war [7]. [4] gives an in-depth overview of recent works in pursuit-evasion differential games. Key relevant trends in this domain identified in the survey include (1) that restriction of information to individual players is a vital area for future research and for accounting for delay of information, (2) in N-pursuer-1-evader games, an interesting solution involves partitioning the space and allocating resources to each space [8], and (3) a subclass of pursuit-evasion games is Active Target Defense Differential Games (ATDDG), as described in the following section.

2.4 Active Target Defense Differential Games (ATDDG)

Active Target Defense Differential Games [4] are a subclass of pursuit-evasion games with an additional defender agent, where the target and defender are on the same team. In this case, the attacker wins if it successfully captures the target, and the target/defender team wins if either the target is never caught or the defender catches the attacker. Often, these games are framed in terms of a missile attacking a ship, where the ship has a defense missile it can launch as well [4].

2.5 Our Work

Our work makes the following contributions:

1. We formulate the essential parts of partially-observable ATDDGs (POATDDGs).
2. We demonstrate how to reason about POATDDGs in the KeYmaeraX framework and formulate and prove winning strategies.
3. We frame a very basic POATDDG in terms of Hybrid Games and discuss ramifications of framing the three-entity game in the two-player framework.
4. We extend the basic game with additional assumptions on ranger mobility or prey behavior, demonstrating how to apply the basic POATTDG game to specific domains.

3 Problem Statement

The basic POATTDG game has three players: poacher (attacker), ranger (defender), and prey (target). However, ranger and prey have the same goal of prey not being captured, so this can very well be framed in terms of the hybrid game. All entities move around a two dimensional space trying to achieve their respective objectives. The game ends when either the poacher kills the prey (the poacher wins) or the ranger catches the poacher (the ranger wins). The ranger also wins if the poacher is never able to kill the prey since the poacher cannot stall the game results forever, and in the edge case that the poacher is caught in the same instant that the poacher kills the prey.

3.1 Incorporating Partial Information

We can incorporate partial information by implementing incompleteness in two types of information: *state observability* and *structural observability*. The former deals with the ability for an entity to observe the rest of the game environment, and the latter deals with the ability of an entity to understand its win conditions and act optimally towards winning the game. In this work, we make poacher powerful by having full state and structural observability. The ranger is less powerful by having full structural observability but only partial state observability. Finally, the prey is generally least powerful by having both partial state observability and partial structural observability.

The goal after formulating the game is to find winning strategies for the ranger, assuming poacher acts optimally. As the prey cannot act optimally, this essentially becomes a two player game between ranger and poacher.

This work focuses on specific applications of partial observability which can be extended to other more specific use cases. We believe that starting from few broad assumptions can make solution strategies more widely-applicable, and as solution strategies are not attainable, more specific assumptions about observability and strength can be added to refine strategies to specific domains. We demonstrate how this can be done by expressing three variants of the POATTDG game with varying assumptions, described at a high-level below.

3.2 Three POATDDGs

We modeled three instances of our hybrid game to learn about the results of having different prey models as well as different information availability. These three games are named after their respective prey models: Panda Game, Elephant Game, and Ostrich Game.

The most basic game is the Panda Game, which models a stationary prey as a panda sitting and eating bamboo. The poacher has access to the entire game state, including locations and velocities of the prey and ranger. In this game, ranger is aware of the prey's location, perhaps via some sensor attached to the prey, but the ranger has no information on the poacher. As the prey does not move, we will find this game to be trivial to reason about but useful for modeling considerations.

The Elephant Game models a slow moving prey as an elephant meandering across a bounded nature preserve. This game involves the same observability level as the Panda Game. This game helps us explore bounds of ranger's requirements when there is no ability to collaborate with an unpredictable prey.

The Ostrich Game models a faster moving prey as an ostrich which runs across the nature preserve, but buries its head in the ground when poachers are near. This means the prey stops moving whenever poachers come within close proximity. In this game, the ranger gains the extra observable information of knowing when the poacher is in close proximity to prey, but the ranger still lacks information on poacher velocity and specific position. The prey gains extra structural information that allows it to react to the poacher in close proximity, and communicate distress to the ranger. We will find this game as an example of adding assumptions to increase collaboration between ranger and prey, thereby decreasing mobility requirements for ranger.

4 Model Description

In this section we formally define the basic aspects of the game. A hybrid game can be defined by its game state, its players, player actions, and win conditions. A skeleton for the way we model the game is provided here on Github. A copy is provided below:

```
Theorem "starlab-dynamics"

Definitions /* CONSTANTS */
  Real maxVp; /* max poacher velocity */
  Real maxVr; /* max ranger velocity */
  Real maxVt; /* max target velocity */

  Real killr; /* how close poacher should be to kill target */
  Real capr; /* how close ranger should be to catch poacher */

  /* functions */
  Real distancesq(Real x1, Real x2, Real y1, Real y2) = (x2-x1)^2 + (
    y2-y1)^2;

  /* Hybrid Programs */
  HP rangerStay ::= {dxR1:=0; dyR1 :=0;}; /* this strategy keeps
    ranger at rest */
  HP rangerGo ::= {dxR1:=dxT1; dyR1:=dyT1}; /* this strategy assumes
    ranger observes prey velocity and copies*/
End.

ProgramVariables /* GAME STATE */
  /* poacher position and velocity */
  Real xP1;
  Real yP1;
  Real dxP1;
  Real dyP1;

  /* ranger position and velocity */
  Real xR1;
  Real yR1;
  Real dxR1;
  Real dyR1;

  /* target position and velocity */
  Real xT1;
  Real yT1;
  Real dxT1;
  Real dyT1;
End.

Problem
  (killr = 0 & maxVt >= 0 & maxVr >= 0 & maxVp >=0 & T>0) /* Initial
    conditions */
  ->
  [
    /* Target chooses initial position (& velocity) */
    xT1 := *; yT1 := *;
    /* Ranger chooses initial position (& velocity) [NOTE: demonic
    choice] */
    {xR1 := *; yR1 := *;}^@;
    /* Poacher position */
    xP1 := *; yP1 := *;
    ?(distancesq(xP1, xT1, yP1, yT1) > killr); /* must start far
    enough away */
    ?(distancesq(xP1, xR1, yP1, yR1) > capr); /* must start far enough
     away */
    {
      /* Target Control */
      dxT1 := *; dyT1 := *; ?(dxT1^2 + dyT1^2 <= maxVt^2);
      /* Ranger Control, generally predefined */
      {rangerStay; ?(dxR1^2 + dyR1^2 <= maxVr^2);}^@;
      /* Poacher Control */
      dxP1 := *; dyP1 := *; ?(dxP1^2 + dyP1^2 <= maxVp^2);

      /* Dynamics */
      {xP1' = dxP1, yP1' = dyP1,
       xR1' = dxR1, yR1' = dyR1,
       xT1' = dxT1, yT1' = dyT1 &
          distancesq(xP1, xR1, yP1, yR1) >= capr^2  &  /* poacher
    caught -> ranger win */
          distancesq(xP1, xT1, yP1, yT1) >= killr^2 &  /* target
    caught -> poacher win */
      };
      ?(distancesq(xP1, xR1, yP1, yR1)> capr^2); /* poacher
    responsible to ensure not caught */
    }* /* poacher can choose to continue the game */
  ]( /* Safety condition */
    distancesq(xP1, xT1, yP1, yT1) > killr^2 /* target never caught */
    )
End.
```

Listing 1: Basic Model

4.1 Players

While ATDDGs are three-entity games, they can be modeled as a two-player game with the target (prey) and defender (ranger) theoretically on the same team. In dGL, the two players are angelic and demonic. In order for the angel player to win, they need to find one path where no matter what demon does, they achieve the goal. The demon player can win if there is no path for the angel to achieve the goal. To this end, while it is possible to model ranger and poacher in either way, it makes more natural sense that ranger be demonic to ensure there is no way for the poacher to achieve their goal of killing the prey.

Interestingly, however, in order to implement partial structural observability of the prey, we will need, in the hybrid game, to place prey on the poacher team. In this way, the ranger team does not control how prey moves, and must account for all possible prey movement. Then we can truly focus on the ranger strategy.

This is made more clear when expressing the model in dGL. Specifically, suppose prey and ranger were both to be on the demonic team, and we wanted to prove there is a strategy for ranger to protect prey. It would be necessary and sufficient to show there exists ranger and prey actions such that the ranger always protects the prey. However, clearly, this is equivalent to asking whether ranger has a strategy assuming prey acts as perfectly as possible. This would be proving something weaker than in a game where the prey has partial structural observability, where they may not act as perfect. By placing the prey on the angelic poacher team, we represent the partial structural observability. Essentially, the prey becomes part of the environment and the ranger must account for all possible prey actions.

To increase prey's structural observability and assume the prey follows some limited behavior in accordance with attempting to win the game, we can put constraints on prey movement, so that it does not collaborate perfectly with poacher. For example, to model a game where the prey moves away from poacher, we can simply force prey to choose actions adhering to the constraint of choosing a velocity away from the poacher.

4.2 Game State

In this work, we focus on the case where there is one poacher, one target, and one ranger in a bounded two-dimensional space. The state of a game at any point in time is the tuple of positions and velocities of the poacher, ranger, and target.

The assumption of a bounded space helps disregard corner cases where if the prey is faster and moves arbitrarily, there will be no way for rangers to keep up with and protect the prey at all points in space without strong assumptions on the poacher or prey mobility and starting position. It is also realistic to our case study as nature preserves are bounded spaces.

4.3 Win Condition

Poacher wins if there is a point in time where the poacher location is equal to the target location without having entered ranger's protection radius, $capr$. Ranger team wins if either the ranger location comes within some fixed range of the poacher location, call this $capr$, or if the poacher can never achieve target location. An example of a winning strategy possible in some games is one where the ranger stands at prey location, so that poacher can never reach the prey without entering the $capr$ radius.

The focus of this project is to find strategies for ranger to win. Then, our desired postcondition for the model involves prey is out of kill range of poacher. We can enforce that poacher does not enter $capr$ range by forcing dynamics to end when poacher enters this range and including a test, as in line 66, directly after dynamics that poacher must satisfy or poacher loses the game.

4.4 Actions and Observations

The model we propose allows target, ranger, and poacher to choose initial positions, where poacher cannot choose the position where the game would immediately end, as controlled by tests in lines 49 and 50.

The rest of the game is a control-dynamics loop, where the poacher may choose whether to continue the loop or to stop the game. First, target, ranger, and poacher may choose their velocities for the next dynamics. Then, the poacher is allowed to control the length of time the dynamics run, subject to the evolution constraints which include constraints that ensure entities stay within our bounded playing space and that the game has not ended.

There are subtle issues in expressing ranger's observation and action space. Line 55 may seem restrictive. It is necessary ranger is not allowed to choose arbitrarily, as target and poacher are able to do with star-assignments. This is because allowing ranger to choose arbitrarily will allow them to make decisions based on both target and poacher state, as they are able to observe target's state in the beginning of the dynamics, and poachers state and the end of the last iteration's dynamics. This will afford ranger the ability to chase poacher, even when ranger is not meant to observe poacher in our game formulation.

Instead, we must define fixed hybrid programs that assign ranger's velocity in terms of the variables they may observe. For example, $rangerStay$ is an example of a strategy ranger may take to stay in place, and $rangerGo$ is an example of a strategy ranger may take to copy the prey velocity. It would also be possible to express that ranger has a choice between multiple simple strategies, but the conditions for ranger to choose one or the other must be predefined in terms of variables ranger can observe. Doing so in KeYmaeraX gives us granular control over accessible game state information.

4.5 Dynamics

We assume motion is only dependent on velocity and do not include acceleration control, as the space in park ranges is large relative to the size and velocity of the entities. Otherwise, our dynamics end on an event-trigger of some end-condition. The game does not continue after one of these events occur.

4.6 Initial State

To begin the game, the prey and ranger can choose an arbitrary starting position within the space. The prey chooses first, and the ranger chooses next. The poacher can choose an arbitrary position that does not end the game by being too close to ranger or prey.

4.7 Assumptions and Limitations

The first limitation is that we stick with the 1-1-1 case, where there is only one of each entity. However, we will find that some strategies naturally allow extension of these cases. These are discussed in Section §7.1.

We also assume a bounded space. This work focuses on faster prey relative to ranger because slow prey relative to ranger creates a trivial solution strategy for ranger, to stay next to the prey. This work focuses on cases where prey is faster than ranger and potentially collaborates very well with poacher. In this case, we find that if the space is unbounded, the prey will be able to run directly toward poacher, and at least far away enough from ranger that the poacher may capture it, assuming poacher can start sufficiently far away.

We also assumed all players get to choose where to start, subject to poacher constraint of not ending the game immediately. This is because we wanted to focus our attention to strategies where ranger is already running the strategy. It is interesting to consider the quickest path to a strategy, but this is likely domain-specific, and not the focus of the paper.

Finally, we originally had modeled so that both the ranger has a range to capture the poacher, and the poacher has a range to kill the prey, but we then decided to make a stronger assumption that poacher must have a kill radius of 0. Most existing pursuit-evasion games do not include any ranges for ending the game and require locations be equal. We make this assumption because the poacher will probably not kill if they cannot gain reward from the prey. In this way, the ranger, in order to fully protect the prey, need only contain the center location of the prey, and not the area around it.

5 Game Proofs

5.1 Proving Winning Strategies for a Demonic Ranger

The game starts with a nondeterministic dual choice for ranger's position. We make use of the fact that dual nondeterministic choice is equivalent to an existence claim. This way, when we reason about the existence of a winning strategy, we directly substitute the strategy to make proofs shorter. This does not affect the soundness of our proof. Namely, if we can prove $\exists x P(x)$, then we can prove $[\{x := *;\}^{@}]P(x)$ in hybrid games.

When we model the system, we force ranger's strategy to be defined in the very beginning in terms of ranger's observations to ensure ranger does not access information such as poacher position or velocity, as described in 4.4. In this way, we prove existence of winning strategies by example.

5.2 Panda Game

We started with the simple case of one stationary prey, one poacher, and one ranger, which is useful for better understanding how to model the game. Below is an abbreviated version of model. We exclude program variables that stay constant throughout all our models.

```
Definitions
  Real maxVp; /* max poacher velocity */
  Real maxVr; /* max ranger velocity */
  Real maxVt; /* max target velocity */

  Real killr; /* how close poacher should be to kill target */
  Real capr; /* how close ranger should be to catch poacher */

  /* functions */
  Real distancesq(Real x1, Real x2, Real y1, Real y2) = (x2-x1)^2 + (
    y2-y1)^2;

  /* Hybrid Programs */
  HP rangerStay ::= {dxR1:=0; dyR1 :=0;};
End.

Problem
  (dxT1 = 0 & dyT1 = 0 & capr >=0 & killr = 0 & maxVp >=0 & maxVr >=0
    & maxVt >=0 &
   xR1 = xT1 & yR1 = yT1 & dxR1 = 0 & dyR1 = 0 & dxR1^2 + dyR1^2 <=
    maxVr^2) /* Initial conditions */
  ->
  [
    /* NOTE:  ranger need not initialize  */
    /* Poacher position */
    xP1 := *; yP1 := *;
    ?(distancesq(xP1, xT1, yP1, yT1) > killr^2);
    ?(distancesq(xP1, xR1, yP1, yR1) > capr^2);
    {
      /* NOTE: ranger and prey choose to not move */
      /* Poacher Control */
      dxP1 := *; dyP1 := *; ?(dxP1^2 + dyP1^2 <= maxVp^2);
      /* Dynamics */
      {xP1' = dxP1, yP1' = dyP1, xR1' = dxR1, yR1' = dyR1, xT1' = dxT1
    , yT1' = dyT1 &
            distancesq(xP1, xR1, yP1, yR1) >= capr^2  &  /* poacher
    caught -> ranger win */
            distancesq(xP1, xT1, yP1, yT1) >= killr^2    /* target
    caught -> poacher win */
      };
      ?(distancesq(xP1, xR1, yP1, yR1)> capr^2);
    }*@invariant(xR1 = xT1 & yR1 = yT1 & distancesq(xP1, xT1, yP1, yT1
    ) > killr^2)
  ]( /* Safety condition */
    distancesq(xP1, xT1, yP1, yT1) > killr^2 /* target never caught */
    )
End.
```

Listing 2: Panda Game Model

5.2.1 Ranger Strategy

The ranger's strategy in this case is to stand on the prey as the poacher cannot kill the prey without coming within the ranger's radius of capture. Then, ranger initially chooses prey position and

5.2.2 Required Assumptions

This strategy works no matter the size of the capture radius for the ranger or max velocities of any entity.

In this model, we also did not require bounded space.

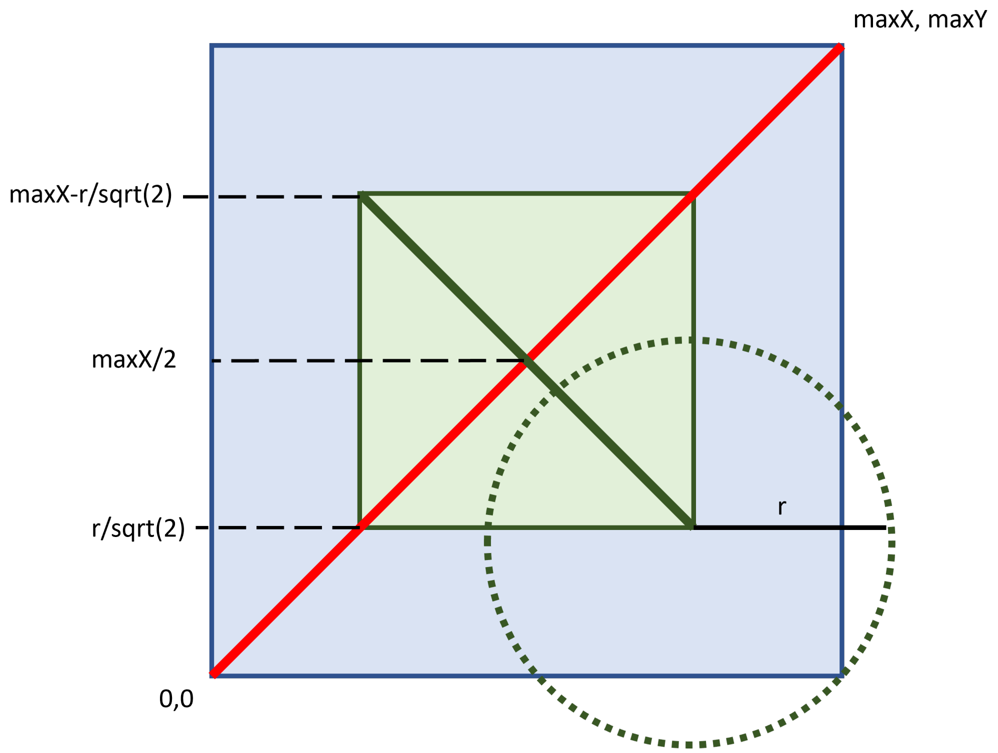

Figure 1: Geometric intuition for the case of one ranger, one poacher, and one prey where all entities may move within a finite space. This figure is a simplified game where the bounded space is square, maxX is the size of the space, and r is equivalent to *capr*.

5.2.3 Proof Intuition

The basic intuition behind the proof involves the invariant that target and ranger position are the same. Therefore, with non-negative *capr*, and given the poacher must start outside that range, the poacher position can never equal the target position and stay outside *capr*.

The proof is quite short and included, along with the full model, here.

5.3 Elephant Game

We then moved on to proving the case with one moving prey, one poacher, and one ranger. Below is an abbreviated version of model. We exclude program variables that stay constant throughout all our models.

```
Definitions
  Real maxX;
  Real maxY;

  Real maxVp; /* max poacher velocity */
  Real maxVr; /* max ranger velocity */
  Real maxVt; /* max target velocity */

  Real killr; /* how close poacher should be to kill target */
  Real capr; /* how close ranger should be to catch poacher */
  Real T; /* ranger shifts */

  /* functions */
  Real distancesq(Real x1, Real x2, Real y1, Real y2) = (x2-x1)^2 + (
    y2-y1)^2;
  Bool vBound(Real dx, Real dy, Real vbound) <-> dx^2 + dy^2 <= vbound
    ^2;
  Bool inBorder(Real x, Real y) <-> 0<=x & x<=maxX & 0<=y & y<=maxY;
  Bool inRangerBorder(Real x, Real y) <-> x >= capr/(2^(1/2)) & y >=
    capr/(2^(1/2)) & x <= maxX-capr/(2^(1/2)) & y <= maxY-capr
    /(2^(1/2));
  Real spaceDiagonal() = (maxX^2 + maxY^2)^(1/2);
  Real rangerDiagonal() = maxX*2^(1/2) - 2*capr; /* assuming maxX =
    maxY */
  Bool poacherDistanceBound(Real x1, Real y1, Real x2, Real y2) <->
    distancesq(x1, x2, y1, y2) > (2*maxVp*T+killr)^2;
     /* want poacher to start far enough away for ranger to initialize
    strategy */
  Bool spaceLowerBound(Real xT1, Real yT1) <-> poacherDistanceBound(0,
     0, xT1, yT1) | poacherDistanceBound(0, maxY, xT1, yT1) |
     poacherDistanceBound(maxX, 0, xT1, yT1) | poacherDistanceBound(
    maxX, maxY, xT1, yT1);
  Real nextTarget(Real pos, Real vel) = pos + vel*T;
  Real spaceToRanger(Real coord, Real max) = capr/(2^(1/2))+coord*(max
    -capr*2^(1/2))/max;


  /* Hybrid Programs */
  /* Predefined Ranger Strategies */
  HP rangerStay ::= {dxR1:=0; dyR1 :=0;};
  HP rangerGo ::= {dxR1:= (spaceToRanger(nextTarget(xT1, dxT1), maxX)-
    xR1)/T;
                  dyR1:= (spaceToRanger(nextTarget(yT1, dyT1), maxY)-
    yR1)/T;};

End.

Problem
  /* Initial conditions */
  (maxX > 0 & maxY > 0 & inBorder(xR1, yR1) & inBorder(xT1, yT1) &
   xR1 = spaceToRanger(xT1, maxX) & yR1 = spaceToRanger(yT1, maxY) &
   killr = 0 & capr >= 0 & maxVp >0 & maxVr >0 & maxVt >0 & T>0 &
   rangerDiagonal()/maxVr <= spaceDiagonal()/maxVt)
  ->
  [
    /* Ranger gets initial position (& velocity) */
    /*{xR1 := *; yR1 := *;
    dxR1 := *; dyR1 := *; ?(dxR1^2 + dyR1^2 <= maxVr^2);}^@;*/
    /* Poacher position */
    xP1 := *; yP1 := *;
    ?(inBorder(xP1, yP1) & distancesq(xP1, xT1, yP1, yT1) > killr^2);
    {
      /* Target control */
      dxT1 := *; dyT1 := *; ?(vBound(dxT1, dyT1, maxVt));
      /* Ranger control */
      rangerGo; ?(vBound(dxR1, dyR1, maxVr));
      /* Poacher Control */
      dxP1 := *; dyP1 := *; ?(vBound(dxP1, dyP1, maxVp));
      /* Dynamics */
      t := 0;
      {xP1' = dxP1, yP1' = dyP1, xR1' = dxR1, yR1' = dyR1, xT1' = dxT1
    , yT1' = dyT1, t' = 1 &
            distancesq(xP1, xR1, yP1, yR1) >= capr^2  &  /* poacher
    caught -> ranger win */
            distancesq(xP1, xT1, yP1, yT1) >= killr^2 &  /* target
    caught -> poacher win */
            t <= T & inBorder(xR1, yR1) & inBorder(xP1, yP1) & inBorder(
    xT1, yT1)
      };
      ?(distancesq(xP1, xR1, yP1, yR1)> capr^2);
    }*@invariant(distancesq(xP1, xT1, yP1, yT1) > killr^2 &
       xR1 = spaceToRanger(xT1, maxX) & yR1 = spaceToRanger(yT1, maxY)
     &
       inBorder(xR1, yR1) & inBorder(xP1, yP1) & inBorder(xT1, yT1))
  ]( /* Safety condition */
    (distancesq(xP1, xT1, yP1, yT1) > killr^2 )/* target never caught
    */ &
    inBorder(xR1, yR1) & inBorder(xP1, yP1) & inBorder(xT1, yT1)
    )
End.
```

Listing 3: Elephant Game Model

5.3.1 Ranger Strategy

The geometric intuition behind the winning strategy is shown in Fig. 1. We find the ranger needs only be able to travel within a subset of the entire space to be able to protect the prey at all points. This subspace, corresponds to the green square in Fig. 1, and depends on the size of ranger's protection radius, $capr$. The bounds of the subspace create a margin of thickness $\frac{capr}{\sqrt{2}}$.

A winning strategy is to stay in lockstep with the prey by taking the prey's location in the total space and finding the corresponding point in the $subspace$ called $projectionPoint$. The ranger will then always be at the $projectionPoint$ for a given prey location, and will always be protecting the prey because of the way $subspace$ is defined.

5.3.2 Required Assumptions

In this case we also limited the area to a finite two dimensional space so that even if the prey moves faster than the ranger, the ranger can keep the prey within range with few velocity and space constraints. The ranger is able to track the target at any point.

Based on the capture radius, we construct $subspace$, and require the constraint that the ranger can travel any distance within $subspace$ in the same time the prey can travel any distance within the entire space. This leads us to need only assume that $\frac{length_{green}}{maxv_{ranger}} \leq \frac{length_{red}}{maxv_{prey}}$.

5.3.3 Proof Intuition

The formal proof, along with the full model .kyx file, is included here.

We have an invariant that the ranger is at $projectionPoint$. Formally, $ranger_x = \frac{capr}{\sqrt{2}} + \frac{maxX - capr\sqrt{2}}{maxX} target_x$, where $ranger_x$ is equal to the x-position of ranger, and $target_x$ is equal to the x-position of target. The same is true for y. We can then prove algebraically that $(ranger_x - target_x)^2 + (ranger_y - target_y)^2 <= capr^2$. This is the basis of our proof. We then know wherever target goes, ranger may follow in safe distance, and because target is always within $capr$, poacher is never able to reach target.

5.4 Ostrich Game

Finally we looked at the case of faster moving prey that freezes when in close proximity $preyr$ to the poacher. In this game, the ranger also now knows when the target sees poacher. Below is a skeletal version of model. We include only new definitions. For the complete model, please see here.

The structure of this game is very different, although the basic dynamics and observability properties are the same. This is because we must split the dynamics into three cases, as there are different ranger strategies for each case. Notably, we include two new ProgramVariables that capture new information the ranger can observe. $tSinceTargetSeenPoacher$ is the length of time since poacher last left the target range. $tSeeingPoacher$ is the amount of time it would take for poacher to catch the target assuming the poacher is in range and moving directly toward the target. As ranger is aware of when target sees poacher and for how long, ranger makes decisions based on these variables.

```
Definitions
  Real preyr; /* distance prey can react to poacher */
```

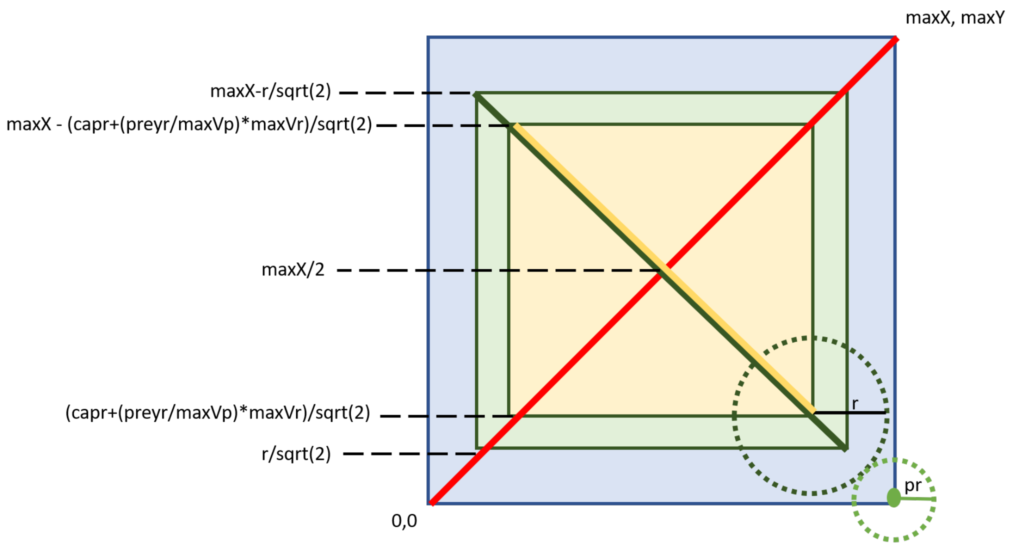
maxX, maxY
maxX-r/sqrt(2)
maxX - (capr+(preyr/maxVp)*maxVr)/sqrt(2)
maxX/2
(capr+(preyr/maxVp)*maxVr)/sqrt(2)
r/sqrt(2)
r
pr
0,0

Figure 2: Geometric intuition for Ostrich Game. Here r is equivalent to $capr$ and $preyr$ is denoted by pr. The yellow square denotes $miniSubspace$ and the green square denotes $subspace$

```

  /* functions */
  Real rangerMiniDiagonal() = maxX*2^(1/2) - 2*(capr+(preyr/maxVp)*
    maxVr);
    /* want poacher to start far enough away for ranger to initialize
    strategy */
  Real spaceToMiniRanger(Real coord, Real max) = (capr+(preyr/maxVp*
    maxVr))/(2^(1/2)) + coord*(max-(capr+(preyr/maxVp)*maxVr)*2^(1/2))
    /max;

  /* Hybrid Programs */
  /*HP targetSeesPoacher ::= {?(distancesq(xP1, xT1, yP1, yT1) <=
    preyr^2);};*/
  /* Predefined Ranger Strategies */
  HP rangerStay ::= {dxR1:=0; dyR1 :=0;};
  HP rangerGo ::= {dxR1:= (spaceToRanger(xT1, maxX)-spaceToMiniRanger(
    xT1, maxX))/(preyr/maxVp);
                   dyR1:= (spaceToRanger(yT1, maxY)-spaceToMiniRanger(
    yT1, maxY))/(preyr/maxVp);
  };
  HP rangerMiniGo ::= {dxR1:= (spaceToMiniRanger(nextTarget(xT1, dxT1)
    , maxX)-xR1)/T;
                   dyR1:= (spaceToMiniRanger(nextTarget(yT1, dyT1),
    maxY)-yR1)/T;};
  HP rangerMiniRetreat ::= {dxR1:= (spaceToMiniRanger(xT1, maxX)-
    spaceToRanger(xT1, maxX))/(preyr/maxVp);
                   dyR1:= (spaceToMiniRanger(yT1, maxY)-spaceToRanger(
    yT1, maxY))/(preyr/maxVp);};
End.

ProgramVariables
  Real tSinceTargetSeenPoacher;
  Real tSeeingPoacher;
End.

Problem
  /* Initial conditions */
  (maxX > 0 & maxY > 0 & inBorder(xR1, yR1) & inBorder(xT1, yT1) &
   xR1 = spaceToMiniRanger(xT1, maxX) & yR1 = spaceToMiniRanger(yT1,
    maxY) &
   killr = 0 & capr >= 0 & preyr > 0 & maxVp >0 & maxVr >0 & maxVt >0
    &
   !targetSeesPoacher(xP1, yP1, xT1, yT1) & tSinceTargetSeenPoacher =
    preyr/maxVp &
   (rangerDiagonal() - rangerMiniDiagonal())/2 <= preyr/maxVp*maxVr &
   (rangerMiniDiagonal())/maxVr <= spaceDiagonal()/maxVt) &
   tSeeingPoacher = 0 & T > 0
  ->
  [
    /* Poacher position */
    xP1 := *; yP1 := *; ?(inBorder(xP1, yP1)); ?(distancesq(xP1, xT1,
    yP1, yT1) > preyr^2);
    {
      {?(tSinceTargetSeenPoacher >= preyr/maxVp); {\alpha;}}
      ++
      {?(tSinceTargetSeenPoacher < preyr/maxVp); {
          {?(targetSeesPoacher(xP1, yP1, xT1, yT1)); {\beta_1;}
          } ++
          {?(distancesq(xT1, xP1, yT1, yP1) >= preyr^2); {\beta_2;}}
      }}
    }*
  ]( /* Safety condition */
    (distancesq(xP1, xT1, yP1, yT1) > killr^2 ) & /* target never
    caught */
    inBorder(xR1, yR1) & inBorder(xP1, yP1) & inBorder(xT1, yT1)
    )
End.
```
Listing 4: Ostrich Game Model Structure

The subgames of alpha, beta_1, and beta_2 are included in the full version of the model. The main differences between these subgames are that ranger has different strategies and $tSinceTargetSeenPoacher$ and $tSeeingPoacher$ are updated accordingly.

5.4.1 Ranger Strategy

The base strategy for ranger in Ostrich Game is similar to Elephant Game. We extend the projection idea to work within a $miniSubspace$, smaller than $subspace$. In this case, we make prey faster. Ranger cannot keep prey protected for the entire game. However, the difference in Ostrich game that allows the loosening of this assumption is that when poacher crosses the prey proximity threshold, $preyr$, the prey stops moving, and ranger knows. The ranger then reacts by moving from target's $miniProjectionPoint$ to target's $projectionPoint$ to protect the prey by the time poacher can kill it. Ranger also reacts to poacher exiting $preyr$ by retreating back to the current $miniProjectionPoint$ so that by the time target starts moving again ranger is in position to stay at $miniProjectionPoint$ for as long as poacher stays outside of $preyr$.

5.4.2 Required Assumptions

Ostrich Game requires three main assumptions. First, we need to assume that ranger can keep up with prey: i.e. ranger can travel across $miniSubspace$ as fast as prey can travel across the entire space. This new assumption is formulated as $\frac{length_{yellow}}{maxv_{ranger}} \leq \frac{length_{red}}{maxv_{prey}}$. We also require an assumption about the relationship between $miniSubspace$ and $subspace$. When poacher comes in close proximity to prey, or in other words enters $preyr$, ranger needs to be able to move from the current $miniProjectionPoint$ to the current $projectionPoint$ by the time poacher kills the prey to make this strategy a winning strategy. This gives us the assumption $\frac{length_{green}-length_{yellow}}{2} \leq catchTime * maxv_{ranger}$ where $catchTime$ is defined as the shortest possible time it takes poacher to go from $preyr$ distance away from prey to killing prey or $\frac{preyr}{maxv_{poacher}}$. The third assumption is assuming that ranger has enough time to retreat back to $miniProjectionPoint$ when poacher leaves $preyr$ before prey starts moving again. The assumption $tSincePreySeenPoacher < catchTime \Rightarrow preyStationary$ where $tSincePreySeenPoacher$

is the time passed since poacher was last within $preyr$, which is implicit in our dynamics, guarantees that property. Lastly, we also require constraining poacher's starting position to be outside of $preyr$.

5.4.3 Proof Intuition

The proof structure will break the three alpha, beta_1, and beta_2 cases into parts. This is necessary because the size of the model and the complexity of the proofs became very large in this case. The game is expressed in the structure simplified structure:

```
preconditions ->
[ {poacher chooses initial position}
  {
    ?(tSinceTargetSeenPoacher >= preyr/maxVp); alpha;
    ?(tSinceTargetSeenPoacher < preyr/maxVp); {
      {?(targetSeesPoacher); beta_1; ++
       ?(distancesq(xT1, xP1, yT1, yP1) >= preyr^2); beta_2; }
    }
  }*@invariant(loop invariants)
](safety conditions)
```

Listing 5: Ostrich Game Model Skeleton

As the crux of the proof resides in the loop step, it suffices to show five subproofs: one for init, one for post, and three for step:

```
// init proof
preconditions -> [ {poacher chooses initial position} ] (loop
    invariants)

// post proof
(loop invariants) -> (safety conditions)

/* STEP Subproofs */
// alpha proof
(loop invariants) -> [alpha](loop invariants)
//beta_1 proof
(loop invariants) -> [beta_1](loop invariants)
// beta_2 proof
(loop invariants) -> [beta_2](loop invariants)
```

Listing 6: Ostrich Subproofs

The loop invariants we end up using are listed below and described in intuitive terms:

```
// We want the poacher to generally be far enough away from the target
     that the ranger can reach the target's projectionPoint by the
    time the poacher reaches the target.  preyr - (tSeeingPoacher*
    maxVp) is the minimum distance poacher will need to travel, which
    we obtain by assuming poacher travels directly to target at max
    velocity.
distancesq(xP1, xT1, yP1, yT1) >= (preyr-(tSeeingPoacher*maxVp))^2
// if tSinceTargetSeenPoacher is large, ranger should be at
    miniProjectionPoint.
(tSinceTargetSeenPoacher >= preyr/maxVp -> (xR1 = spaceToMiniRanger(
    xT1, maxX) & yR1 = spaceToMiniRanger(yT1, maxY)))
// we want ranger to be at the average of projectionPoint and
    miniProjectionPoint, weighted by tSeeingPoacher
(xR1 = ((tSeeingPoacher/(preyr/maxVp))*spaceToRanger(xT1, maxX) + (1-(
    tSeeingPoacher/(preyr/maxVp)))*spaceToMiniRanger(xT1, maxX)))
(yR1 = ((tSeeingPoacher/(preyr/maxVp))*spaceToRanger(yT1, maxY) + (1-(
    tSeeingPoacher/(preyr/maxVp)))*spaceToMiniRanger(yT1, maxY)))
// tSeeingPoacher is equivalent to the amount of time we have to get
    to miniProjectionPoint
(0 <= tSeeingPoacher & tSeeingPoacher <= preyr/maxVp)
// tSinceTargetSeenPoacher is equivalent to the amount of time passed
    since target stopped seeing poacher
```

```
(0 <= tSinceTargetSeenPoacher & tSinceTargetSeenPoacher <= preyr/maxVp
    )
// tSeeingPoacher is upperbound by preyr/maxVp - the amount of time
    since target stopped seeing poacher
(tSeeingPoacher <= preyr/maxVp - tSinceTargetSeenPoacher)
// if tSinceTargetSeenPoacher is large, target does not see poacher.
    This loop invariant is important to transfer information across
    cases
(tSinceTargetSeenPoacher >= preyr/maxVp -> !targetSeesPoacher(xP1, yP1
    , xT1, yT1))
// if targetSeesPoacher, we know tSinceTargetSeenPoacher = 0.  This
    loop invariant is important to transfer information across cases.
(targetSeesPoacher(xP1, yP1, xT1, yT1)->tSinceTargetSeenPoacher = 0)
// basic boundary invariant
inBorder(xR1, yR1) & inBorder(xP1, yP1) & inBorder(xT1, yT1)
```

Listing 7: Ostrich Game Loop Invariants

The init and post subproofs are trivial. The full proof of these subgames are included in the init and post .kyp files.

In the alpha case, we only care that we are at the miniProjection point at all times in this case. This is very similar in structure to the Elephant Game proof, as described in 5.3. The full proof of the alpha subgame is included , which was proven before attempting to break down the proof.

```
?(tSinceTargetSeenPoacher < preyr/maxVp); {
{?(targetSeesPoacher(xP1, yP1, xT1, yT1));
/* Target control STOPS when sees poacher */
dxT1 := 0; dyT1 := 0; ?(vBound(dxT1, dyT1, maxVt));
/* Ranger control */
{{?(tSeeingPoacher=preyr/maxVp); rangerStay;}
++
{?(!(tSeeingPoacher=preyr/maxVp)); rangerGo;}}
?(vBound(dxR1, dyR1, maxVr));
/* Poacher Control */
dxP1 := *; dyP1 := *; ?(vBound(dxP1, dyP1, maxVp));
/* Dynamics */
{{xP1' = dxP1, yP1' = dyP1, xR1' = dxR1, yR1' = dyR1, xT1' = dxT1, yT1
    ' = dyT1, tSeeingPoacher' = 1 &
distancesq(xP1, xR1, yP1, yR1) >= capr^2  &  /* poacher caught ->
    ranger win */
distancesq(xP1, xT1, yP1, yT1) >= killr^2 &  /* target caught ->
    poacher win */
inBorder(xR1, yR1) & inBorder(xP1, yP1) & inBorder(xT1, yT1) &
targetSeesPoacher(xP1, yP1, xT1, yT1) & tSeeingPoacher <= preyr/maxVp
}
++
{xP1' = dxP1, yP1' = dyP1, xR1' = dxR1, yR1' = dyR1, xT1' = dxT1, yT1'
     = dyT1 &
distancesq(xP1, xR1, yP1, yR1) >= capr^2  &  /* poacher caught ->
    ranger win */
distancesq(xP1, xT1, yP1, yT1) >= killr^2 &  /* target caught ->
    poacher win */
inBorder(xR1, yR1) & inBorder(xP1, yP1) & inBorder(xT1, yT1) &
targetSeesPoacher(xP1, yP1, xT1, yT1) & tSeeingPoacher >= preyr/maxVp
    }};
?(distancesq(xP1, xR1, yP1, yR1)> capr^2);
}
}
```

Listing 8: Ostrich beta_1 Subgame

In the beta_1 case, poacher is within $preyr$ of target. This means, ranger must move towards the $projectionPoint$ and stay there if we are at that point. This breaks up the proof into two parts: one where the ranger moves, and one where the ranger does not move. Further, the dynamics are broken into two more parts: $tSeeingPoacher <= preyr/maxVp$ and $tSeeingPoacher >= preyr/maxVp$. We case on each of these:

1. ranger stops and dynamics $tSeeingPoacher <= preyr/maxVp$: This is an easy trivial case because $tSeeingPoacher$ must be equal to $preyr/maxVp$, as per the test. Then, the dynamics don't move, and all invariants hold.
2. ranger stops and dynamics $tSeeingPoacher >= preyr/maxVp$: This is also fairly trivial because $tSeeingPoacher$ ranger and target both do not move, and ranger is at $projectionPoint$. We need only prove the safety that distance between poacher and target is safe, which is true because we know ranger at $projectionPoint$ means ranger protects target.
3. ranger moves and $tSeeingPoacher >= preyr/maxVp$: This is a trivial because $0 <= tSeeingPoacher < preyr/maxVp$, as per the test, which means the poacher fails the test of the evolution domain constraint.
4. ranger moves and $tSeeingPoacher <= preyr/maxVp$: This is the non-trivial step. However, we will find that most loop invariants are trivial. We need show ranger is at the correct weighted average, which proves by dI and invariance of target position, as we know projection points remain constant throughout the dynamics. The other nontrivial loop invariant is the first safety, which requires proving that the poacher will always be as far from the prey as ranger is from $projectionPoint$. We were not able to prove this, but believe it follows directly from the intuition that straight, direct paths are shortest paths.

The full proof of the beta_1 subgame is included here. Note, we proved this with a weaker, unprovable loop invariant $distancesq(xP1, xT1, yP1, yT1) > killr()^2$. The beta_2 subgame is symmetric.

6 Challenges

While modeling and proving our hybrid game, we ran into a few challenges worth discussing.

6.1 Strategy Complexity

One of the biggest challenges in our project was strategy complexity. We would come up with what appeared to be a winning strategy for ranger, however, it was not good enough if it did not translate into a dGL proof. One of the ways in which we managed to create strategies that were clean enough to translate into proofs was to think geometrically and guarantee more specific properties. For example, we used our $projectionPoint$ to have an invariant relation between ranger and prey positions. We also broke Ostrich Game down into subcomponents depending on the stage of the game (whether poacher was near prey recently or not).

6.2 Gazelle Game

Originally, before we decided on modeling Ostrich Game, we had planned to do Gazelle Game. The difference is that in Gazelle Game, instead of freezing when poacher comes within close proximity, the prey runs away. However, shortly into our strategy discussions we ran into a few obstacles. Our first idea was to base our strategy on keeping some amount of overlap between $preyr$ and $capr$ to make sure poacher could not cut directly between prey and ranger as without assuming ranger to be faster than poacher, a concession we did not want to make, we would be lost in that case. However, this led into a couple of issues. First, our assumptions and invariants had to deal with reasoning about the overlap of $preyr$ and $capr$, and well as the intersection points between $preyr$ and $capr$ which would make proving the model in KeyMaeraX much more difficult since we had a limited timeline. The second issue was guaranteeing that ranger did not end up lagging more and more behind prey as the game progressed. Without our $projectionPoint$ and $miniProjectionPoint$ strategies from Elephant Game and Ostrich Game, we could not show that ranger would be able to keep up with prey for every possible movement choice prey might choose.

7 Future Work

7.1 Extensions to Multiple Entities

We intend to show based on this result that there is a relationship between our velocity and space constraints and the number of rangers (i.e. if we double the space and double the number of rangers, they still have a winning strategy if each ranger takes responsibility of half of the new space).

7.2 New Environmental Factors

Secondly, we want to add terrain throughout the space. This terrain will slow the poacher and ranger uniformly, but not the prey. This will require further creativity from the poacher and ranger to decide what angle to chase the prey from, and which areas of the playing space to focus on.

References

[1] André Platzer. Differential game logic. *ACM Trans. Comput. Log.*, 17(1):1:1–1:51, 2015.

[2] André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, Cham, 2018.

[3] Rufus Isaacs. Differential games: Their scope, nature, and future. *J. Optim. Theory Appl.*, 3(5):283–295, May 1969.

[4] Isaac E. Weintraub, Meir Pachter, and Eloy Garcia. An introduction to pursuit-evasion differential games, 2020.

[5] F. Fang, T. Nguyen, Arunesh Sinha, Shahrzad Gholami, Andrew Plumptre, L. Joppa, Milind Tambe, Margaret Driciru, F. Wanyama, Rob Critchlow, and Colin Beale. Predicting poaching for wildlife protection. *IBM Journal of Research and Development*, 61:3:1–3:12, 11 2017.

[6] Timothy Kuiper, Blessing Kavhu, Nobesuthu A. Ngwenya, Roseline Mandisodza-Chikerema, and E.J. Milner-Gulland. Rangers and modellers collaborate to build and evaluate spatial models of african elephant poaching. *Biological Conservation*, 243:108486, 2020.

[7] Y.-C Ho, A.E. Bryson, and S. Baron. Differential games and optimal pursuit-evasion strategies. *Automatic Control, IEEE Transactions on*, AC10:385 – 389, 11 1965.

[8] Efstathios Bakolas and Panagiotis Tsiotras. Optimal pursuit of moving targets using dynamic voronoi diagrams. pages 7431 – 7436, 01 2011.