Platforming Games: Formalization and Properties

JESSICA PAQUETTE, CSC 482A

1 INTRODUCTION

A *platforming game* is a type of video game that was made most famous by Nintendo's *Super Mario Bros.* in 1985. The player's objective is to navigate a preset stage, starting at a single preset point s, and ending at a preset point t. The stage consists of *platforms* which the player may jump onto, and obstacles that the player must avoid. Some games may employ a time-limit, a point system, or other additions, but at its core, all that is required is the player, platforms, and obstacles.

Platforming games, in the general case, are at least PSPACE-hard [1]. PSPACE-hard games include the *Donkey Kong Country*. Other games, such as the two-dimensional *Mario* games are NP-hard.

Due to their overall popularity, platforming games have enjoyed significant interest in the game AI community. The yearly *Platformer AI Competition* challenges members of the community to create effective game-solvers and level-generators [2]. In the 2009 competition, Robin Baumgarten created a level-solver using the A* pathfinding algorithm. This level-solver, although impressive, has several weaknesses, and as will be shown, reduces to the Canadian traveler problem.

2 BACKGROUND

2.1 Platforming Games

As stated in the introduction, a *platforming game* is a game which requires a player to traverse a stage starting from a preset point s to a preset point t without dying in the process. The ways in which the player can die are defined from game to game. Most common are falling in pits, and being injured by an obstacle.

Mario is a platforming game in which the player starts, in general but not always, on the lefthand side of the screen, and traverses the stage in order to reach the goal on the rightmost side of the stage. Mario also features power-ups which add more complexity to the solution of the game; if Mario has a power-up, he may be able to take a certain path through a level that he would not be able to otherwise. It is also worth noting that several levels in many Mario games may have more than one goal point.

There are two popular ways to measure the quality of a solution to a stage in Mario: *minimum time* and *maximum score*. Minimum-time solutions require the player to simply traverse the stage from the starting position to the goal in the least time possible. Maximum score solutions are more complex, as there are many game-constructs that have a positive impact on player score. Such

constructs include collecting coins and power-ups, defeating enemies, and completing the stage in the minimal amount of time.

The general platforming problem is at least PSPACE-hard by reduction from PushPush-1 [1]. No proof has been constructed to show whether or not the problem is EXPTIME-hard.

2.2 The Canadian Traveller Problem

The Canadian Traveller Problem (CTP) is a PSPACE-hard problem that generalizes the shortest path problem on graphs without perfect information. The player learns more about the graph's structure by exploring it. Although CTP generalizes shortest *path*, it is important to note that the best strategy may not be a path at all [3].

The graph that the player wishes traverse can take many forms. Each of these forms is called a *realization*. The player wishes to find a *policy*, a strategy that can be used on any realization of a CTP instance. Any such policy results in a different walk for the individual realizations of a CTP instance. CTP asks the expected cost of the optimal policies.

Formally, the problem consists of a family of graphs $\mathcal{G}$, a set E of edges that must be in the current realization, and a set F of edges that *may* be in the current realization. We define a cost matrix W whose entries w_{uv} denote the cost of travelling from the vertex u to the vertex v, given that the edge uv is in the current realization. A strategy that allows the player to traverse the graph from s to t is a mapping from $(\mathcal{P}(E), \mathcal{P}(F), V)$ to V. The cost of some strategy is the sum of the weights of the edges the strategy visits.

2.3 A* Pathfinding

A* is a pathfinding algorithm that extends Dijkstra's algorithm by using a heuristic. Such heuristics include Manhattan distance and Euclidean distance. A*'s objective is, for every vertex in the search graph, to minimize

$$f(n) = g(n) + h(n)$$

Where $g(n)$ is the *total distance* from s, as described by Dijkstra's algorithm, and $h(n)$ is the estimated distance from the current vertex to t. At every iteration of the algorithm, A* chooses the smallest f value. If, at any point during the algorithm A* detects a vertex with a smaller $g(n)$ value, the vertex is marked as unvisited. The behaviour of A* depends on the influence the heuristic has on the decisionmaking process. If it has no influence, A* becomes Dijkstra's algorithm. If it entirely decides the course of action, then A* becomes the Greedy Best-First Search algorithm.

Implementation requires two lists: `OPEN` and `CLOSED`. `OPEN` consists of those vertices that have not been visited, and `CLOSED` consists of those that have been visited or cannot be added to the path [4].

3 FORMALIZATION

The simplest platforming problem can be described as the pair

$$(L, P)$$

Where L denotes the stage or level, and P denotes the player. However, in order to obtain a sufficient formalization, we must describe the stage, player, and game state for a platforming game.

3.1 Defining the Platformer Stage

A *stage* or *level* is a quintuple

$$L = (x_L, y_L, x, y, O_L, P_L, G_L)$$

The values $x_L \in \mathbb{Z}^+$ and $y_L \in \mathbb{Z}^+$ denote the total width and height of the stage. Similarly, $x \in \mathbb{Z}^+$ and $y \in \mathbb{Z}^+$ denote the *screenwidth* and *screenheight* of the stage at any point; screenwidth and screenheight bound the space that is observable to the player at any given time. The set O_L refers to the obstacles that exist in L, and the set P_L refers to the platforms that exist in L. Finally, the set G_L contains the goalpoints in the stage.

Note that $x \leq x_L$ and $y \leq y_L$, as the screenwidth and screenheight must remain within the level's boundaries. Additionally, the game is in polynomial time if $x = x_L$ and $y = y_L$ [1].

3.1.1 Obstacles

The set O_L consists of triples of the form (o, x_o, y_o), where o is the obstacle, and x_o and y_o refer to the obstacle's location at the game's current state. Many platforming games feature nonstationary obstacles, and if they are featured, the game state must also account for the state of the obstacle set. If an obstacle is neutralized or killed, and will not return to the game, it is removed from O_L. If the player's x_p and y_p values match the x_o and y_o values of any triplet in O_L, then the player either sustains damage or dies.

The definition of P_L is similar, but instead it contains the location of any platforms in the game.

3.1.2 Goals

G_L contains x_t, y_t pairs for every goal-point $t \in G_L$. If the player's x_p and y_p values match any pair contained in G_L, then the game is over and the player wins. Note that G_L cannot be empty. Similar games can be defined where G_L does not exist, but they shall be considered variants of the platforming problem. The Mario AI Championship and Platformer AI Competition feature such games.

3.2 Defining the Platformer Player

Next, we define the player P. The player, controlled by the user, is allowed to move left, right, stay still, or jump. The game state must keep track of what the player is doing at any point in time, and keep track of whether or not the player is alive, or dead. If the player ever enters the dead state, the game is over. Then,

$$P = (x, y, d \in \{0, 1\}, B)$$

defines the player.

The variables $x \in \mathbb{Z}^+$ and $y \in \mathbb{Z}^+$ describe the player location and provide a bridge between the state of the player and the state of the stage. When the player's x value and y value reach a certain threshold, the screen the player is on must be updated.

Player mortality is described by d. If $d = 0$, then the player is alive. If $d = 1$ the player is dead. The game ends when the player is dead.

The set B encodes what the player is doing at any given point in the game. B contains every legal action the player may take. A simple construction may contain binary values which encode "walk right", "walk left", "jump right", "jump left" and "stay still."

3.3 Defining the Game State

The state of the game must encapsulate

- The current instance of the stage L
- The current state of the player P
- The current screen

The stage instance is handled by its definition; all that may change in the stage during the game is the number of obstacles and the number of platforms. Similarly, the player's state is handled by its definition. The current screen must be determined from the player's x and y coordinates and the dimensions of the stage.

The way this is handled depends on the game construct. Mario handles the current screen by only updating the current screen to ensure that the player is always in the centre. That is, the number of screens in the game is determined by the number of half-screenwidths.

4 PLATFORMING USING A*

In 2009, Robin Baumgarten entered the 2009 Mario AI Competition [5] and won using an A*-based agent. This win was controversial, as Baumgarten copied the game's physics engine to allow the agent to determine the optimal path to take. Although this solution played Mario incredibly well, it does not overcome the hardness of the problem.

4.1 Baumgarten's A* Platforming Algorithm

As previously stated, A* is a pathfinding algorithm that extends Dijkstra's algorithm with use of a heuristic. Baumgarten's Mario AI estimated the best path by first looking for the choice that would allow the player to reach the rightmost side of the screen as quickly as possible, and then deciding whether or not that course of action would injure the player. If the best path would injure Mario, the algorithm would backtrack through the search graph and choose the next best path. In order to find the next best path, Baumgarten copied the game physics engine in order to estimate the game state [6].

At every iteration in the game, the player state is described as the current node (the player's current location) and the nodes that the player would reach by performing any legal action in the game. Whenever an action is performed, if the current node has not been visited, a new state must be created.

4.2 Shortcomings of Baumgarten's Solution

Baumgarten's solution was restricted to a very small, very convenient subset of the platforming problem based off of Markus Persson's *Infinite Super Mario Bros.* [7].

Platforming games exist within a virtualized continuous space, due to jumping physics. Additionally, the running and walking mechanic used in Mario allow the player to "skid" or "slide" on the ground. This makes it unlikely that the player will return to a node already contained in the search graph [6].

Additionally, stages in platforming games often feature levels that force the character to move backward, through the means of dead-ends or puzzles. The search-space built up by Baumgarten's AI makes it infeasible for it to handle such levels [6].

Finally, Baumgarten's AI is not guaranteed to find the *best* path in the game using its heuristic. It may be faster to take a path that would hurt Mario when he is in a state where he can sustain damage (certain items in Mario allow him to survive contact with an obstacle).

5 BAUMGARTEN'S SOLUTION AND CTP

Baumgarten's solution can be described as an instance of CTP. Recall that the Canadian Traveller Problem (CTP) is a generalization of the shortest path problem on graphs without perfect information.

The stage L is a lattice with $x_L \cdot y_L$ vertices. Initially, the edge set E that contains the edges that *must* be in the solution consists of the edges that the A* algorithm visits and does not associate a penalty with. The edge set F contains the remaining edges in the stage L. The weight matrix W associates a very high cost with edges that lead to an obstacle, and the cost the heuristic computes with every other edge.

The player does not know the cost of travelling between two nodes u and v until the current node is u and it is adjacent to v.

The player's objective is to find a policy, that no matter what it does, it will always find a good solution. That is, the edge set E contains only edges that will yield a reasonable solution that is no more than c times greater than the optimal solution.

Through this simple construction, we can see that Baumgarten's solution is actually an instance of CTP.

6 CONCLUSION

Platforming problems can be formalized in terms of set-theoretical constructs. Handling such problems is difficult due to their (virtually) continuous nature. AI solutions exist that solve the problem very effectively under a restricted subset. Every Mario game has an associated CTP instance via Robin Baumgarten's A*-based solution. The problem continues to be of interest in the platforming community, and could offer students a lighthearted introduction to hard problems.

REFERENCES

[1] G. Aloupis, E. D. Demaine, and A. Guo, "Classic nintendo games are (np-)hard," *CoRR*, vol. abs/1203.1895, 2012.

[2] "Platformer ai competition," http://www.platformersai.com/.

[3] "Canadian traveller problem," http://en.wikipedia.org/wiki/Canadian_traveller_problem, updated: 2014-05-06.

[4] "Amit's a* pages," http://theory.stanford.edu/ amitp/GameProgramming, updated: 2014-10-04.

[5] "Mario ai competition 2009," http://julian.togelius.com/mariocompetition2009/.

[6] "Infinite mario ai using a* search: The definitive interview with robin baumgarten," http://aigamedev.com/open/interviews/mario-ai/.

[7] J. K. Julian Togelius, Sergey Karakovskiy and J. Schmidhuber, "Super mario evolution," 2009.