Real-Time Navigation in Classical Platform Games via Skill Reuse

Michael Dann **Fabio Zambetta** **John Thangarajah**
School of Science, RMIT University, Australia
{michael.dann, fabio.zambetta, john.thangarajah}@rmit.edu.au

Abstract

In platform videogames, players are frequently tasked with solving medium-term navigation problems in order to gather items or powerups. Artificial agents must generally obtain some form of direct experience before they can solve such tasks. Experience is gained either through training runs, or by exploiting knowledge of the game's physics to generate detailed simulations. Human players, on the other hand, seem to look ahead in high-level, abstract steps. Motivated by human play, we introduce an approach that leverages not only abstract "skills", but also knowledge of what those skills can and cannot achieve. We apply this approach to *Infinite Mario*, where despite facing randomly generated, maze-like levels, our agent is capable of deriving complex plans in real-time, without relying on perfect knowledge of the game's physics.

1 Introduction

In platform videogames, levels are typically completed by travelling to the right of screen. Therefore, a common approach taken by artificial agents is to perform a low-level forward search, guided by a heuristic that rewards rightward movement [Togelius *et al.*, 2010; Jacobsen *et al.*, 2014]. Unfortunately, this approach has several drawbacks: Firstly, it requires knowledge of the game's *forward model*, that is, a model of how the environment will evolve given any course of action. Most existing agents derive this model directly from the game's code, which is "cheating" in a sense. Secondly, searching at a granular level limits the lookahead depth achievable in real-time. Lastly, obtaining items and powerups often requires navigation around obstacles (see Figure 1), for which a "move towards target" heuristic may lead the agent astray. In this paper, we propose an alternative method that looks ahead at a higher level and does not require an exact forward model. On maze-like navigation problems in *Infinite Mario*, it finds complex plans in real-time and significantly outperforms a state-of-the-art low-level search agent.

Human videogame players appear to contend with granular time increments by acquiring extended "skills", such as *running* and *jumping*. Skills, together with the "god's eye view" afforded in most platform games, allow human players to visualise ahead in abstract, high-level steps. For example, the result of a *jump to platform* skill can be visualised by imagining the protagonist tranferred to the target platform. This type of visualisation allows human players to solve new problems upon inspection, i.e. before actually setting out.

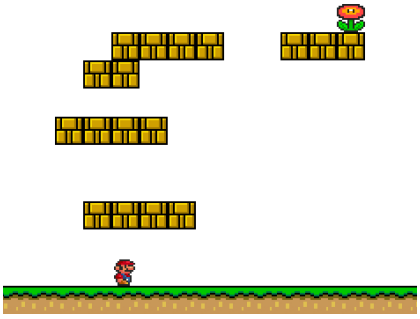

Figure 1: The "fire flower" at the top-right of screen cannot be reached by following a direct path.

While skill acquisition has been an active area of research within machine learning for some time [Thrun *et al.*, 1995; Digney, 1998; Menache *et al.*, 2002; Pickett and Barto, 2002; Mannor *et al.*, 2004; Şimşek and Barto, 2004; Şimşek *et al.*, 2005; Konidaris and Barto, 2009; Vezhnevets *et al.*, 2016], the common approach of associating skills with "bottlenecks" (states that must be passed through in order to navigate from one distinct region of the state space to another) requires the analysis of many sample trajectories per task instance. As such, it is ill-suited to solving new problems upon inspection. Furthermore, the bottleneck method usually only identifies task-specific skills. For example, it might identify a skill for jumping to the specific platform above Mario in Figure 1, but not a general "jump to platform" skill. The ability of human players to solve new problems in real-time seemingly relies on the latter type of skill.

To this end, the first contribution of this work is an approach for acquiring transferable skills that does not rely on upfront identification of bottlenecks. We make a distinction between composite behaviour (e.g. *jumping then running then jumping*) and fundamental "basis" behaviour (i.e. the separate behaviours of *jumping* and *running*), and propose a reward scheme for isolating the latter.

Our second contribution is a method for deriving skill-based plans in real-time. Under previous approaches, skill transfer has the potential to *speed up* learning on new problems [Konidaris and Barto, 2007], but some task-specific experience is still required to learn a high-level plan. Under our approach, plans are derived through transferred knowledge of skills' capabilities and limitations; for example, the fact that it is possible to jump x units high, but that it is impossible to run through walls. Consequently, bottlenecks fall out at the *end* of the process; a reversal of the traditional order.

We title our overall approach *Synoptic Vision Planning (SVP)* to reflect its target domain: problems that afford the agent a "god's eye view" of the world. Like hierarchical learning methods [Dietterich, 2000; Parr and Russell, 1998; Sutton *et al.*, 1999] and hierarchical search techniques [Vien and Toussaint, 2015], SVP decomposes long-term tasks into subtasks. However, unlike past approaches, SVP does not rely on task-specific training runs or low-level simulations. Instead, it leverages transferred skill knowledge and god's eye vision to project high-level plans.

2 Background & Related Work

In this section we review some past approaches to platform videogames, then provide a brief overview of skill-related learning methods, highlighting the gap that SVP addresses.

2.1 Existing Platform Videogame Agents

Platform videogames gained attention as a domain for artificial intelligence research with the advent of the *Mario AI Competition* in 2009 [Togelius *et al.*, 2010]. The competition centred around the game *Infinite Mario*, an adaption of *Super Mario Bros.* with a built-in random level generator. To date, the most successful *Infinite Mario* agents have been based on search methods such as *A** and *MCTS* [Jacobsen *et al.*, 2014]. However, these agents are best-suited to "flat" levels, requiring significant streamlining to achieve even a few seconds of planning depth [Jacobsen *et al.*, 2014]. They also generally rely on exact knowledge of the game's forward model.

Recently, *Reinforcement Learning (RL)* [Sutton and Barto, 1998] has gained popularity as an alternative approach to videogame AI. This is largely due to the breakthrough work of Mnih *et al.* [Mnih *et al.*, 2015], who trained an *Atari 2600* agent to play many games to human level from raw pixel input alone. Impressive as this was, their agent also struggled on medium-term navigation tasks. On the adventure game *Montezuma's Revenge*, their agent was unable to make any progress even after 50 million training frames. Very recent work has seen significant progress in this game, with the agent of Bellemare *et al.* [Bellemare *et al.*, 2016] learning to reach 15 out of 24 rooms after 50 million frames. However, it essentially relied on brute force exploration, maintaining a memory and striving to reach novel states. There is no suggestion by the authors that their agent transfers to new rooms without further training.

2.2 Skills

Humans appear to cope with fine-grained time increments by acquiring extended "skills". In this work we model skills via the *Options Framework* [Sutton *et al.*, 1999], which is perhaps the most popular approach. However, we note that alternative skill models exist, such as *MAXQ* [Dietterich, 2000] and *Hierarchical Abstract Machines (HAMs)* [Parr and Russell, 1998]. Informally, an option may be thought of as a skill that is only applicable in certain situations. Formally, an option, o, is defined as a tuple $\langle I, \pi, \beta \rangle$ where:

- $I \subseteq S$ is the set of states the option can be initiated from.
- $\pi : S \times A \rightarrow [0,1]$ is a policy that returns the probability of selecting action a when in state s.
- $\beta : S \rightarrow [0,1]$ returns the probability that the option will terminate in a given state.

A key idea under all frameworks cited above is that skills can be composed to facilitate hierarchical learning. For example, a high-level *get fire flower* skill might call two lower level skills, *run* and *jump*. This is typically done to increase learning efficiency. Providing the agent with useful extended actions may effectively reduce the planning depth required [Sutton *et al.*, 1999]. However, even in the case where low-level skills can be transferred, the top level policy must be retrained for each new problem instance. Games where the level structure is randomised at the start of each episode, such as *Infinite Mario*, do not afford this opportunity. Therefore, we seek an alternative approach to deriving skill-based plans that does not require task-specific training runs.

3 Synoptic Vision Planning (SVP)

In this section we introduce our approach, *Synoptic Vision Planning (SVP)*. SVP consists of three components:

- First, the agent learns a transferable skill for performing local movement (Section 3.1).
- The agent then learns a general estimator for the likelihood that a local movement will succeed (Section 3.2).
- Faced with a longer term planning task, the agent leverages "god's eye vision" to project paths composed of local movements. This is illustrated schematically in Figure 2. The path with the greatest probability of success is found via Dijkstra's algorithm by weighting each step with its log-likelihood of success (Section 3.3).

Since SVP is probabilistic, there is some chance that a plan may fail. Our approach to identifying plan failure is described in Section 3.4.

3.1 Training Local Movement

To train a local movement policy, we define a tiling over the environment's spatial dimensions and teach the agent how to navigate to nearby tiles. Formally, this is modelled as follows:

- Let $\psi : S \rightarrow T$ be a tiling that maps each state $s \in S$ to a tile $t \in T$, based on the protagonist's spatial coordinates. The tile size controls the precision of the agent's high-level movements. For classical platform games, it is natural to match this to the game's grid size.
- Let $d : T \times T \rightarrow \mathbb{R}^+ \cup \{0\}$ be a distance metric over the set of tiles. In this work, we use the Manhattan distance between tile centroids.

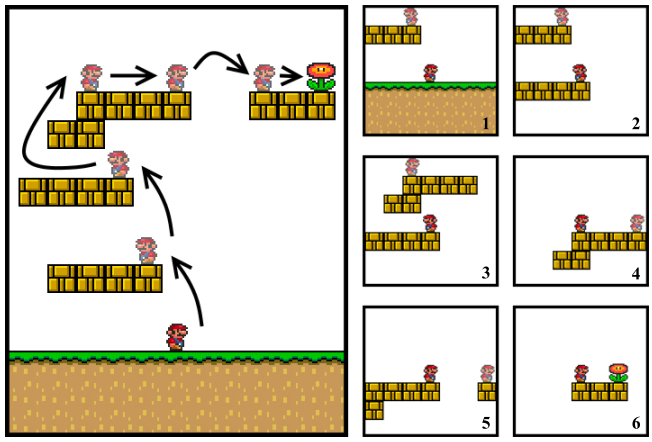

Figure 2: A schematic illustration of our skill-based lookahead, using an example from *Infinite Mario*.

- Given tile t and a constant $D \in \mathbb{R}$, we define t's *set of neighbouring tiles* $N_t^D \subseteq T$ as

$$N_t^D = \{t' \in T \mid d(t,t') \leq D, t' \neq t\}$$

  Under a Manhattan metric, N_t^D corresponds to a square grid centred on t (see Figure 3). The *neighbourhood size parameter*, D, controls the reach of the local movement policy. For platform games, we suggest that this be set to the protagonist's maximum jump distance.

- At the start of each training episode, we designate a random tile within N_t^D (where t is the protagonist's current tile) as the *training goal*. It is assumed that training goals can placed over a variety of scenarios, and that the state representation is *agent-centric* [Konidaris and Barto, 2007] so that the policy learned is transferable.

Given this setting, two possible reward schemes seem natural: either a binary reward of +1 for reaching the training goal and zero elsewhere, or an incremental reward for reducing distance to the training goal. In Section 5 we evaluate these schemes empirically and provide some intuitive reasons as to why the latter leads to better overall performance.

For any two tiles t_a and t_b such that $t_b \in N_{t_a}^D$, the trained policy induces a natural skill for navigating from t_a to t_b.

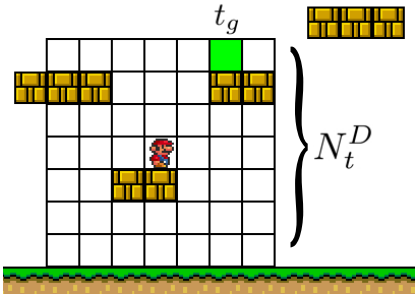

Figure 3: Local movement is trained by rewarding the agent for reaching a training goal, $t_g \in N_t^D$.

Following the formalism introduced in the background, we model this skill as an option $o_a^b = \langle I_a, \pi_b, \beta_b \rangle$ where:

- $I_a = \psi^{-1}(t_a)$
- π_b is a policy for navigating to t_b, derived by setting the local movement policy's goal equal to t_b.
- $\beta_b(s) = \begin{cases} 0, & \text{if } s \notin \psi^{-1}(t_b) \\ 1, & \text{if } s \in \psi^{-1}(t_b) \end{cases}$

3.2 Predicting the Likelihood of Success of a Skill

Rather than relying on a precise forward model, we take a high-level, probabilistic approach to predicting skill outcomes. After the local movement policy has been trained, we continue setting random training goals. We set a time limit, T_{max}, and observe whether the local movement policy succeeds within this limit. This generates a series of samples from which the following function can be approximated:

$$\Pr(o_a^b, T_{max}|s) \equiv Pr(o_a^b \text{ terminates within } T_{max} \text{ time steps when initiated from } s \in I_a)$$

Konidaris and Barto [Konidaris and Barto, 2009] train a probability estimator in similar fashion, but their estimator is trained for a fixed task. Our estimator is trained over a variety of conditions such that, ideally, it will learn the type of high-level game mechanics that a human player understands. For example, in *Infinite Mario*, it should learn to estimate low probabilities if t_a and t_b are separated by a wall.

3.3 Planning by Exploiting Synoptic Vision

Referring back to Figure 2, the above estimator allows the agent to calculate the likelihood of Step 1 succeeding, since the details of the current state, s, are known. From this point, human players are capable of looking further ahead by exploiting the "god's eye" or *synoptic* view afforded in platform videogames. A human can visualise the result of Step 1 succeeding (indicated by the first transparent Mario), and thus estimate the likelihood of further steps succeeding (Step 2 being one possible continuation).

An immediate difficulty that arises in mirroring this approach is that Step 1's termination state is uncertain. Mario is unlikely to arrive stationary at the exact centre of Step 1's target tile. However, if we make the reasonable assumption that the likelihood of the next step succeeding will not be greatly affected by small differences in its initiation state then a crude estimate may suffice for planning. Accordingly, we construct a *hypothetical initiation state,* $\hat{s}_t$, by assuming that the protagonist will arrive at the exact centre of the previous step's target tile, t, with zero velocity. More sophisticated approaches are certainly possible and are the subject of future work.

Hypothetical initiation states allow us to estimate the log-likelihood of a sequence of steps succeeding, by summing the log-likelihoods of the individual steps:

$$\begin{aligned} \textit{Log-Likelihood}_{seq} = -(&\log[\Pr(o_a^b, T_{max}|s)] \\ &+ \log[\Pr(o_b^c, T_{max}|\hat{s}_{t_b})] \\ &+ \log[\Pr(o_c^d, T_{max}|\hat{s}_{t_c})] + \cdots) \end{aligned}$$

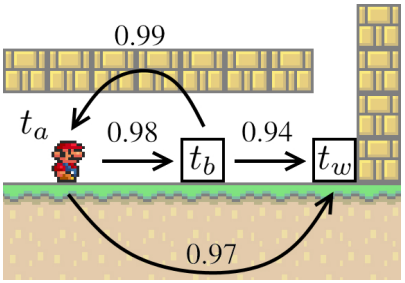

Figure 4: An example from *Infinite Mario* where the agent will become stuck trying to reach the next waypoint, t_w.

Observe that the current state, s, is used to calculate the probability of the first step succeeding, while hypothetical initiation states are used thereafter. Given an arbitrary goal tile on the current screen, we calculate the plan with the greatest likelihood of success by treating log-likelihoods as path lengths and applying Dijkstra's algorithm.

3.4 Replanning Threshold

Since our planning framework is probabilistic, and so in general are option policies, there is no guarantee that a plan will succeed. The agent may fail a step during execution, e.g. Mario may fall from a ledge so that the next waypoint is no longer within local movement range. Therefore, it will sometimes be necessary to replan.

A straightforward approach to replanning is to just recalculate the entire plan every frame. If the optimal path changes, the agent will automatically adjust. This is similar in essence to interrupting an option if its action-value drops below that of an alternative action [Sutton *et al.*, 1999, Section 4]. However, under our probabilistic approach, continual replanning may cause problems if the probability estimates are not entirely consistent, as in Figure 4. In this example, the agent has determined that the next waypoint is t_w. However, on the way to t_w it encounters t_b and estimates a lower likelihood of reaching t_w than it did originally, despite having moved closer.[1] At this point the agent redirects to t_a, having estimated a greater chance of success for $t_b \rightarrow t_a \rightarrow t_w$ than $t_b \rightarrow t_w$. Hence the agent retreats and becomes stuck running between t_a and t_b.

To address this issue, our approach is to replan only if the current step is deemed failed. This occurs iff:

$$t_w \notin N^D_{t_b} \quad \text{or} \quad min[\frac{\Pr(o^w_b, T_{max}|s_b)}{\Pr(o^w_a, T_{max}|s_a)}, 1] \leq k$$

where t_w is the target waypoint, s_b and t_b are the current state and tile, s_a and t_a are the state and tile from which the step was initiated, and $k \geq 0$ is the *replanning threshold*.

The first condition ensures that the step is failed if the target is no longer in local movement range. The logic behind the second condition is as follows: The current likelihood of reaching t_w is compared to the original estimate. If the current estimate is significantly lower, it stands to reason that something has gone wrong during execution. A relative rather than an absolute threshold is used because the current step may have had a low chance of succeeding to begin with. (It may be an intrinsically difficult step). Setting $k = 1$ is equivalent to continual replanning, while $k = 0$ means that the plan will only be calculated if the next waypoint falls out of range. In principle, the parameter should be set low enough that cycles are avoided, but high enough that the plan will be reset if the agent makes a clear error. For the example in Figure 4, the cycle will be avoided so long as $k < \frac{0.94}{0.97}$.

[1] The example here is fictional, but this type of phenomenon did arise in our experiments. Due to episodes where the training goal was placed on the other side of a wall, the agent may have learned to associate wall proximity with low success likelihood.

4 Experimental Configuration

We evaluated SVP on randomly generated navigation puzzles in *Infinite Mario*. In this section, we describe the evaluation task, the configuration of the learning and planning components of SVP for *Infinite Mario*, and the benchmark used.

4.1 Evaluation Task

To generate the type of maze-like tasks that have proven difficult for artificial agents to date, we modified the game's level generator to create structures of the type shown in Figure 5. At the start of each episode, a reachable, non-mid-air tile from a band 9 – 11 tiles away from Mario was randomly assigned as the goal. Agents were provided with a view that extended 5 tiles beyond this band for tasks that necessitated travelling past the goal then backtracking. In the interests of evaluating navigation ability only, enemies were disabled.

4.2 Training Configuration

We trained two types of local movement policy to compare the two reward schemes (binary and incremental) mentioned near the end of Section 3.1. For the incremental scheme, the precise reward form used was:

$$R(s_t, s_{t+1}) = \Phi(s_t) - \Phi(s_{t+1})$$
$$\Phi(s) = Distance(\text{protagonist, goal})$$

Under this scheme, the "directness" of the resultant policy is controlled by the discount factor, γ. With $\gamma = 0$, the agent is trained to move greedily in the direction of the training goal. With $\gamma > 0$ some level of indirectness is tolerated, which is necessary for training curved movements such as jumps. In our experiments, we used $\gamma = 0.7$. Since the resultant policy broadly favours straight movement over complex

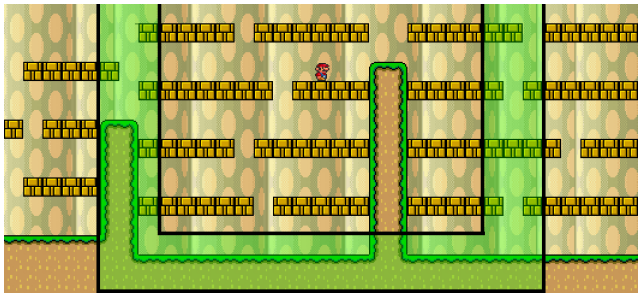

Figure 5: The goal placement zone for our experiments.

Learning rate	2.5×10^{-4}	Momentum	0.95
Action length	4 frames	TD error sep.	3 actions
Experience cache size	1.5×10^{6} frames	Min. cache pop. for train	$50,000$ frames
Exploration policy	ϵ-greedy	ϵ decay schedule	Hyperbolic $1.0 \rightarrow 0.05$
Target net refresh rate	$10,000$ frames	Training time	2×10^{8} frames

Table 1: The training parameters used for *Infinite Mario*.

manoeuvres involving direction changes, we refer to it as the *basis* movement policy, by analogy with vector terminology.

For the binary reward scheme, a much milder discount was required to ensure that the reward for reaching the training goal was not effectively hidden. We used $\gamma = 0.98$ in our experiments. This scheme favours flexible local movement by whichever route is fastest. Hence, we refer to the corresponding policy as the *flexi* movement policy.

The tile size was set equal to that of a brick, which is a natural configuration for many classical 2D videogames. The neighbourhood size was set equal to Mario's maximum jump height, which is 5 bricks.

The state representation used was very similar to that of Togelius *et al.* [Togelius *et al.*, 2009]. It contained Mario's velocity, the training goal position and a binary encoding of brick positions within Mario's neighbourhood. Since this encoding rounds brick positions to the nearest tile, we also included Mario's fractional offset from the grid centre.

For the training algorithm we used *Q-Learning* [Sutton and Barto, 1998, Section 6.5]. Action-values were approximated via a fully connected, feed forward architecture with 2 hidden layers each containing 300 neurons. We also employed the stability measures of Mnih *et al.* [Mnih *et al.*, 2015], using *experience replay* [Lin, 1993] and maintaining separate training and target networks, updating the target network periodically. Other training parameters were guided by Mnih *et al.*, then tuned to the values in Table 1 by hand.

4.3 Planner Configuration

The skill success probability estimator was trained via supervised learning, with a time limit of $T_{max} = 5$ seconds given to complete local movements. We used the same network architecture as above, but for stability we reduced the learning rate to 2.5×10^{-5} and increased training time to 8×10^{8} frames. We saved connection weights intermittently and took the network with the lowest MSE.

For Dijkstra's algorithm, we configured the search space to be the full set of tiles visible on current screen. For efficiency, we overrode the distance metric such that $d(t_1, t_2) = \infty$ if t_2 is in mid-air or part of a wall, effectively disabling all connections to such tiles. Mid-air and mid-wall locations are not natural waypoints in the vast majority of platform games, so including them would potentially waste a significant amount of time, both in training and planning.

The replanning threshold, k, was tuned for each policy by hand. For the basis movement policy, the overall performance of SVP was strongest with a relatively low replanning threshold of $k = 0.2$. For the flexi movement policy, a value of $k = 0.4$ performed best.

4.4 Benchmark

We benchmarked our approach against Robin Baumgarten's A* agent, which won the *2009 Mario AI Competition* [Togelius *et al.*, 2010]. Since *Infinite Mario* requires agents to select an action every 40ms, it is usually not possible for A* to run to completion. This problem persists despite a number of performance hacks in Baumgarten's approach, such as not expanding the current node if Mario is at a similar location at a similar time in a pre-existing node. In cases where the search time runs out, it is necessary to select the most promising incomplete branch via a heuristic. For "flat" levels, Baumgarten's agent chooses the branch that extends furthest to the right of screen. To repurpose the agent for point-to-point navigation, we modified it to select the branch terminating closest to the goal, in terms of Euclidean distance. Unfortunately, in maze-like environments it is difficult to estimate the "true" distance remaining without solving the maze upfront. This is why the evaluation task is non-trivial despite A*'s theoretical guarantees when given enough time and memory.

5 Results

We conducted two experiments. In the first, we compared the *basis* and *flexi* versions of SVP against the A* benchmark. Next, we examined the impact of varying the replanning threshold. All success rates were calculated over 10,000 evaluation episodes per agent.

5.1 SVP versus A*

As Figure 6 illustrates, both versions of SVP achieved far greater success rates than A* in the long run. A* reached more goals over the first few seconds because, for the problems it could solve, it calculated near optimal paths by exploiting the game's exact forward model. However, for tasks where its heuristic led it to a dead-end, the A* agent became permanently stuck. By contrast, the SVP agents' low-level

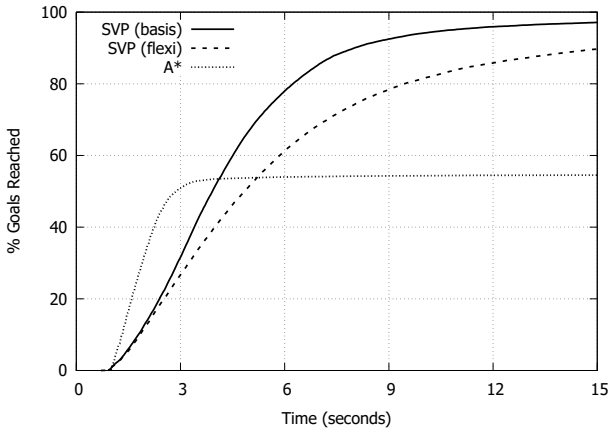

Figure 6: Success rate versus time taken for the *basis* and *flexi* versions of SVP and the A* benchmark.

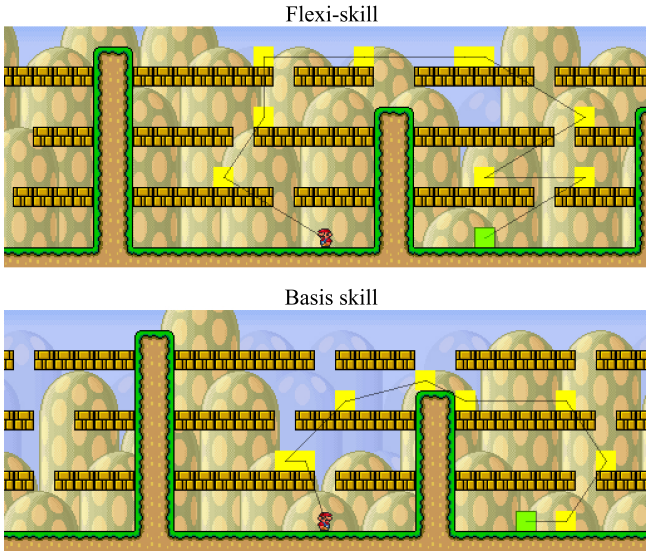

Figure 7: The basis skill agent tended to identify simpler, more "natural" waypoints than the flexi-skill agent. The goal is highlighted green, the waypoints determined are highlighted yellow.

execution was not perfect, but they generally found viable plans and were able to recover from mistakes.

Interestingly, even though the *flexi* policy was more versatile than the *basis* policy (by the end of training, it was reaching around 95% of training goals, versus only 77% for the *basis* policy), the *basis* policy was stronger in conjunction with the planner. The *basis* agent's success rate on the evaluation task was 97.1% (±0.2%), versus only 90.0% (±0.3%) for the *flexi* agent. While it appears in Figure 6 that the *flexi* agent may eventually catch up, completion times in excess of 15 seconds are arguably unreasonable for the type of task generated. We did in fact try extending the limit to 30 seconds, but the *basis* agent still led, 98.6% (±0.1%) to 95.1% (±0.2%).

While the *basis* policy itself could not perform complex manoeuvres, the agent's planner was able to segment complex steps into direct movements. The *flexi* agent was less constrained in its waypoint placement because its local movement policy was able to tolerate some planning burden. However, this meant that the *flexi* agent's paths were often less efficient. Figure 7 provides an illustration of this effect. An additional advantage of the basis movement approach is that the likelihood-of-success estimator does not have to recognise complex paths; it only has to recognise whether a direct path to the next waypoint is possible.

5.2 Replanning Threshold

In our second experiment we studied the effect of varying the replanning threshold, k. We compared the optimal value for the basis skill agent ($k = 0.2$) against continual replanning ($k = 1.0$) and replanning only when the next waypoint became out of range ($k = 0.0$).

Referring to Figure 8, the difference between the $k = 0.2$ and $k = 0.0$ lines has a straightforward interpretation. The $k = 0.0$ agent reached 85.4% (±0.4%) of goals within 15 seconds, indicating that step failure was rare. However, it could

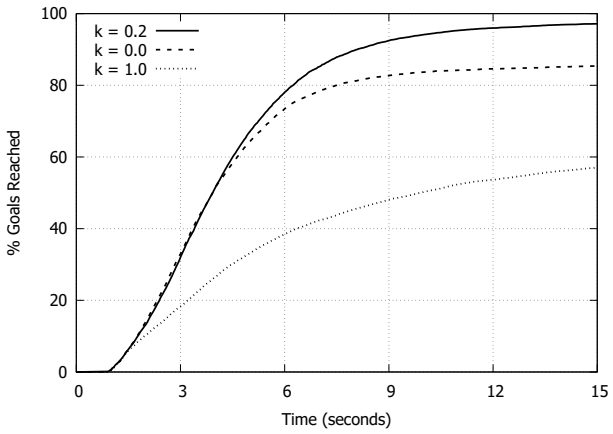

Figure 8: The *basis* agent's success rate versus time taken for different values of the replanning threshold, k.

not recover when steps did fail, while the $k = 0.2$ agent could, albeit completing such episodes slowly. Hence, the advantage of the $k = 0.2$ agent only became pronounced towards the right of the graph.

Continual replanning was severely detrimental, with the $k = 1.0$ agent only reaching 57.2% (±0.5%) of goals within 15 seconds. It appears that there were often many viable plans with similar success probabilities. Small changes in the protagonist's position and velocity were often enough to alter the plan rankings, causing the agent to switch plans frequently without making any progress. The fact that the optimal replanning threshold was so low ($k = 0.2$) also suggests that probability estimates varied significantly during execution, such that it was only worth replanning when the agent had high confidence that the current step had failed.

6 Conclusion

In this paper we introduced *Synoptic Vision Planning (SVP)*, a real-time, skill-based approach to navigating domains where a "god's eye view" is provided but the low-level dynamics are unknown. We evaluated SVP in *Infinite Mario* and showed that it was capable of solving complex navigation tasks in real-time, far outperforming an A* agent that exploited the game's exact forward model.

SVP's lookahead method relies critically on the provision of a "god's eye" or synoptic view. However, these perspectives are relatively common, occurring not only in games, but also in many real-world problems, such as a rover operating with satellite overhead. Nonetheless, besides SVP, we are not aware of any artificial approaches that explicitly leverage synoptic representations for planning.

In its current form, SVP is best-suited to classical 2D games where the game world is laid out according to a grid. Besides *Infinite Mario*, other examples include *Metroid*, *Alex Kidd in Miracle World* and *Gauntlet*. To apply SVP to less structured environments there are some challenges that must be overcome, such as handling the case where a tile is partially blocked by an obstacle. We plan to address this and extend SVP to more complex domains in future work.

References

[Bellemare *et al.*, 2016] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[Dietterich, 2000] Thomas G Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 13:227–303, 2000.

[Digney, 1998] Bruce L Digney. Learning Hierarchical Control Structures for Multiple Tasks and Changing Environments. In *Proceedings of the 5th International Conference on Simulation of Adaptive Behavior*, volume 5, pages 321–330, 1998.

[Jacobsen *et al.*, 2014] Emil Juul Jacobsen, Rasmus Greve, and Julian Togelius. Monte Mario: Platforming with MCTS. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, pages 293–300. ACM, 2014.

[Konidaris and Barto, 2007] George Konidaris and Andrew Barto. Building Portable Options: Skill Transfer in Reinforcement Learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 895–900, 2007.

[Konidaris and Barto, 2009] George Konidaris and Andrew Barto. Skill Discovery in Continuous Reinforcement Learning Domains Using Skill Chaining. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1015–1023, 2009.

[Lin, 1993] Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, 1993. Technical Report CMU-CS-93-103.

[Mannor *et al.*, 2004] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic Abstraction in Reinforcement Learning via Clustering. In *Proceedings of the 21st International Conference on Machine Learning*, page 71. ACM, 2004.

[Menache *et al.*, 2002] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning. In *Machine Learning: ECML 2002*, pages 295–306. Springer, 2002.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-Level Control Through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.

[Parr and Russell, 1998] Ronald Parr and Stuart Russell. Reinforcement Learning with Hierarchies of Machines. *Advances in Neural Information Processing Systems (NIPS)*, pages 1043–1049, 1998.

[Pickett and Barto, 2002] Marc Pickett and Andrew G Barto. Policyblocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. In *Proceedings of the 19th International Conference on Machine Learning*, volume 2, pages 506–513, 2002.

[Şimşek and Barto, 2004] Özgür Şimşek and Andrew G Barto. Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning. In *Proceedings of the 21st International Conference on Machine Learning*, page 95. ACM, 2004.

[Şimşek *et al.*, 2005] Özgür Şimşek, Alicia P Wolfe, and Andrew G Barto. Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 816–823. ACM, 2005.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*, volume 1. Cambridge Univ Press, 1998.

[Sutton *et al.*, 1999] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1):181–211, 1999.

[Thrun *et al.*, 1995] Sebastian Thrun, Anton Schwartz, et al. Finding Structure in Reinforcement Learning. *Advances in Neural Information Processing Systems (NIPS)*, pages 385–392, 1995.

[Togelius *et al.*, 2009] Julian Togelius, Sergey Karakovskiy, Jan Koutník, and Jürgen Schmidhuber. Super Mario evolution. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 156–161. IEEE, 2009.

[Togelius *et al.*, 2010] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. The 2009 Mario AI Competition. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[Vezhnevets *et al.*, 2016] Alexander Vezhnevets, Volodymyr Mnih, Simon Osindero, Alex Graves, Oriol Vinyals, John Agapiou, et al. Strategic Attentive Writer for Learning Macro-Actions. In *Advances in Neural Information Processing Systems (NIPS 2016)*, pages 3486–3494, 2016.

[Vien and Toussaint, 2015] Ngo Anh Vien and Marc Toussaint. Hierarchical Monte-Carlo Planning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 3613–3619, 2015.