Flow as a Metacontrol for AI Agents

by

Thorey Maria Mariusdottir

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

Abstract

Flow is a psychological state that provides a person with enjoyment when their skills and the challenges of their activity match. Thus flow encourages people to persist at and return to just manageable challenges; thereby fostering the growth of skills over time. It seems natural to assume that flow is an evolutionary adaption because there may be an advantage from using each individual's survival skills to the fullest in a hostile environment. This work introduces a method for metacontrol in artificial intelligence based on the concept flow. We propose that if an artificial intelligent agent is guided to move through tasks in a way that balances their skills with the complexity of the task this may improve its performance. To this end we define a measure of flow that allows agents to seek the state closest to flow. This measure is based on the reciprocal of the absolute difference between an agent's skills and the complexity of the task. The complexity is learned through observing, at each point, the minimum skills required to complete the task. We first illustrate our approach using a simple model of a multi-level environment and then compare our approach to a scripted and random metacontrols in the video game of *Angband*. The results indicate that a flow-seeking metacontrol can show improvement over a random metacontrol but the scripted metacontrol performs better overall.

Acknowledgements

Thanks to my supervisor Vadim Bulitko for his help and support in guiding me through this thesis and to Matthew Brown for offering invaluable insight. Thanks also to the IRCL research group for all their help and interesting discussions.

I am also grateful to all my friends at the University of Alberta for offering support and making my experience here enjoyable.

Finally, my eternal gratitude as always goes to my parents, Helga and Marius; my sister, Elin; and my brother, Johann. Their support has been essential and I would not be where I am today without them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Traditionally within artificial intelligence (AI) decision-making is modelled with an action-perception loop similar to that shown in Figure 1.1. An AI agent observes the state of the environment and selects an action that allows it to achieve its goal. Then it perceives the result of that action and the cycle continues.

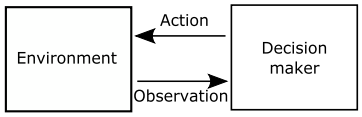

Figure 1.1: The action-perception loop model of a decision-making process.

Nevertheless, significant research exists on the idea that intelligence may also require agents to think about their own thinking [2, 9]. *Metacontrol* (or metareasoning) is the monitoring and control of the decision-making cycle. It represents a higher layer of control, that perceives and acts on the whole decision-making process rather than just the environment (Figure 1.2). The subject of this thesis is a method for metacontrol in artificial intelligence based on *flow*.

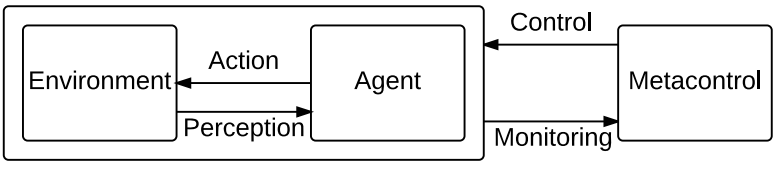

Figure 1.2: The metacontrol of a decision-making process.

Csikszentmihalyi [10] defined flow as the subjective state that people report when they are so immersed in an activity that they lose awareness of time, worries,

and even their sense of self. A key characteristic of Csikszentmihalyi's model of flow is the concept of matching skills and difficulty. He identified three regions of experience: boredom, if an activity is too easy; anxiety, if the difficulty exceeds skills; and the flow state, where skill and difficulty are matched.

The state of flow is autotelic: it provides a person with enjoyment that comes not from outside rewards but from just being in this state. These experiential rewards encourage a person to persist at and return to just manageable challenges; thereby fostering the growth of skills over time. This is supported by studies that associate flow with commitment and achievement during the high school years [8, 29, 32]. Thus seeking flow becomes a form of metacontrol guiding people towards optimal use of their skills; where they are neither underqualified nor overqualified.

1.1 A Motivating Example

To illustrate consider the role-playing game of *Angband* [11]. *Angband* is a dungeon exploration game divided into a 100 levels. Figure 1.3 shows a snapshot of a typical level. The player starts on level 0 and each successive level is guarded by more formidable monsters. She must defeat the monsters to clear each level but as she does, she gains new skills and becomes a better fighter. The object of the game is to reach the 100^{th} level and defeat Morgoth, the Lord of Darkness. However, if the player dies before that he must start again from the beginning.

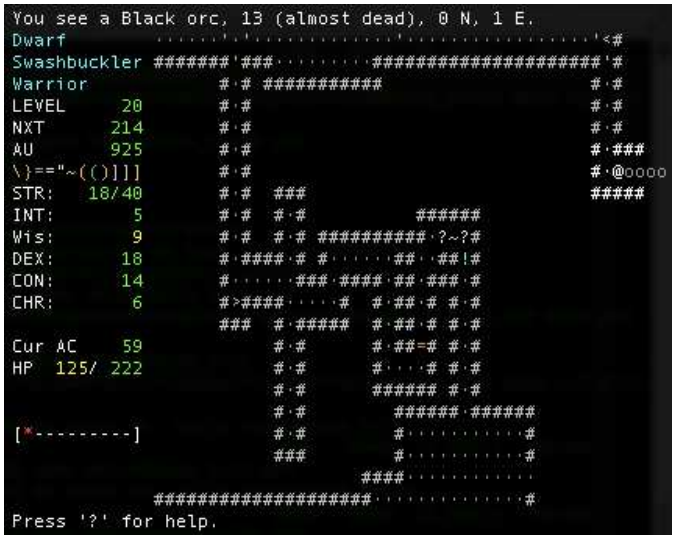

Figure 1.3: A screen-shot from *Angband*.

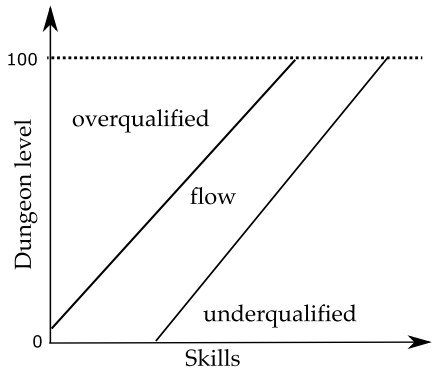

Figure 1.4: The state of flow in *Angband*.

Figure 1.4 demonstrates the dynamics of flow in *Angband*. When the dungeon level is high while at the same time the skills of the player are low, the player may feel anxiety or frustration. On the other hand, when the level is low while the skills are high; the player may become bored. Between the boredom and frustration lies the state of flow where level and skills are commensurate.

If we define metacontrol in *Angband* to be the monitoring of decision-making on each level and control of level progression, then seeking flow is one such metacontrol. Seeking flow causes players to make certain decisions that are based not only on the current game state but on their state and their performance in the game. If they are having trouble with the monsters they may become anxious and slow down their level progression or return to a previous level; but if the monsters present no challenge boredom makes them progress faster through the levels. At the same time flow offers intrinsic rewards when the progression is just right.

The game of *Angband* is an example of an environment where we speculate that seeking flow can be beneficial. When the player dies, the score given is based on experience points. Experience points are gained for each monster killed and are higher the greater a challenge the monster is expected to be for the character. Thus, the faster a player progresses through the levels the more experience points they will gain per kill, but the risk of dying also increases if the player has not gained sufficient skills. Flow will guide the player to take on the most challenging monster their skills can manage; thus accumulating the most experience points they can without increasing the risk of death through lack of qualification.

For an AI agent playing *Angband* it may be useful to introduce a similar *flow-seeking metacontrol*. This could help balance the decision-making between the two disparate goals of avoiding death and collecting experience points. The goal of avoiding death is best achieved by maintaining a progress slow enough that the player remains continuously overqualified. However, this results in few experience points scored. On the other hand, the most experience points are collected by advancing levels fast to take on high-level monsters but then the probability of dying is high. A flow-seeking metacontrol balances the two goals as the agent considers its own performance in the game and stays at a level it has the skills to survive but is not overqualified for.

1.2 Thesis Contributions

This thesis contains two main contributions. Our first contribution is a method for metacontrol in level-based environments based on flow. Chapter 2 provides a formal description of level-based environments and the problem of metacontrol within them and Chapter 3 reviews related methods of metacontrol applicable to the problem. Informally, a level-based environment is one where the problem of navigating from an initial state to a goal state requires traversing a set of levels of varying complexity. Our proposed method controls level progression by matching the skills of the agent to the perceived complexity of the level. To do so in we introduce a mathematical model quantifying degree of flow and propose an algorithm for metacontrol that focuses on maximizing this degree of flow (Chapter 4).

We propose that this flow-seeking metacontrol improves performance in level-based environments. To this end the second contribution of this thesis is an empirical study in *Angband* (Chapter 5) to compare our method to other metacontrols that are unaware of flow: a random policy and an established scripted player for *Angband*. Chapter 6 discusses the strengths and shortcomings of our method along with directions for future work.

Chapter 2

Problem Definition

This work considers sequential decision-making process in an environment that decomposes into levels of varying difficulty. In particular, we focus on one part of the problem: controlling level progression of the agent. This chapter offers a mathematical formulation of the level-based environment and the metacontrol problem this thesis aims to explore.

2.1 Level-based Environments

We frame the problem in the traditional agent-environment interface where the decision maker is called the *agent* and everything outside the agent is called the *environment*.

We consider an agent performing sequential decision-making in a stochastic environment with a given start state and goal state. We can model this as a stochastic decision-making process, defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{S}^\dagger, s_0 \rangle$. Here $\mathcal{S}$ is a finite set of states; $\mathcal{A}$ is a finite set of actions; $\mathcal{P}(s'|s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is a transition probability function that gives the probability of moving from state s to state s' by executing action a; $\mathcal{S}^\dagger$ is a set of absorbing terminal states that includes the goal state, s_g; and s_0 is the start state.

At first the environment is in state $s_0 \in \mathcal{S}$. At each time t the agent selects an action, $a_t \in \mathcal{A}$, and the environment transitions from the current state, s_t, to a new state, s_{t+1}. The objective of the agent is to navigate from the initial state, s_0, to a goal state s_g, in as few steps as possible. Reaching any other state in $\mathcal{S}^\dagger$ constitutes a failure.

We will consider only environments that have levels of varying complexity (Figure 2.1). Following the notation of Bulitko [6] we assume there exists a sequence of *levels*, $\{L_i\}_{i=0}^{N} \subset \mathcal{S}$, in the state space such that:

(i) The set of states in a level, L_l, $l \in \{0, ..., N\}$, is a non-empty subset of $\mathcal{S}$ and $s_0 \in L_0$. For all $0 \leq i < j \leq N$ we also have $L_i \cap L_j = \emptyset$. For convenience we define $l(s)$ to be level index of a state $s \in \mathcal{S} \setminus \mathcal{S}^{\dagger}$.

(ii) There exist exactly two terminal states in $\mathcal{S}^{\dagger}$: a *goal state*, $s_g \in L_N$, and a *death state*, $s_d \notin \bigcup_{i=0}^{N} L_i$.

(iii) The state space includes only the states of each level and the death state: $S = \{s_d\} \cup \bigcup_{i=0}^{N} L_i$

(iv) The set of all actions, $\mathcal{A}$, can be divided into two disjoint sets of actions: the set of all *inter-level actions*, A, and the set of all *intra-level actions*, $\tilde{A}$.

(v) For each inter-level action, $a \in A$, there exists exactly one *intended level*, l_a, such that $\mathcal{P}(s'|s, a) > 0$ only if $s' \in L_{l_a} \cup \{s, s_d\}$. In short, an inter-level action, a, can only move the agent to the intended level l_a or keep it in the current state unless it dies. In addition, if $l_a = l(s)$ then $\mathcal{P}(s'|s, a) = 0$ if $s' \notin \{s, s_d\}$ (i.e., inter-level actions do not change the state within a level).

(vi) For each level, $l \in \{0, ..., N\}$, there exists a non-empty set of *intra-level actions*, $\tilde{A}_l \subset \tilde{A}$. An intra-level action, $\tilde{a} \in \tilde{A}_l$, will never change the level: $\mathcal{P}(s'|s, \tilde{a}) = 0$ if $l(s') \neq l(s)$.

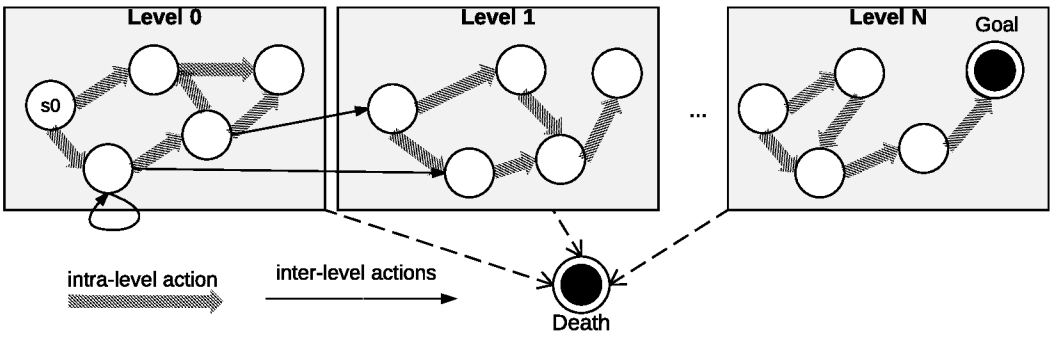

Figure 2.1: Organization of states and actions in a level-based environment.

The policy of an agent is a mapping from the states to actions. Bulitko [6] proposed that, for level-based environments, we decompose the policy of the agent into two parts: the *ground policy*, $\tilde{\pi}$, is a policy restricted to intra-level actions: $\tilde{\pi} : \mathcal{S} \to \tilde{A}$; and the *metacontrol policy*, π, is a policy restricted to inter-level actions, $\pi : \mathcal{S} \to A$.

Note that the state space of these two policies need not be the same since their environment is not. The general rule, as set forth by Sutton [42], is that anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment. The agent-environment boundary is meant to represent the limit of the agent's absolute control. In that sense, the environment of the metacontrol will be different from that of the ground policy. In particular, unless the ground policy is under the absolute control of the metacontrol, the ground policy must constitute a part of the metacontrol's environment. For simplicity, however, we propose that the metacontrol and ground policy operate on different representations of the same state space $\mathcal{S}$.

The two policies separate the control of the agent into two parts: control and metacontrol (Figure 2.2). The ground policy is acting only within the current level, but will never take an action that brings it to a different level. On the other hand, the metacontrol policy can only keep the agent in the same state or move it to a state on another level.

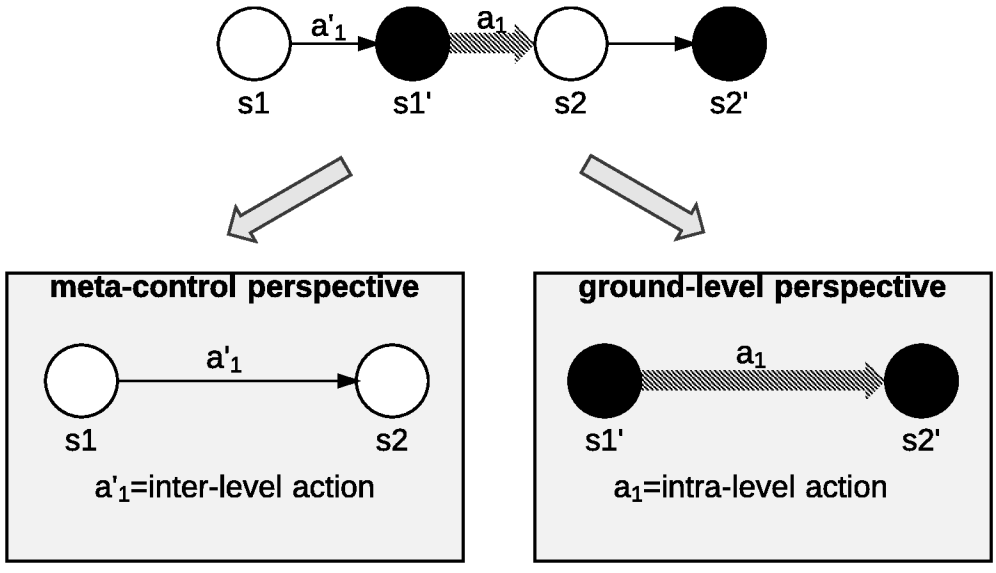

Figure 2.2: Decomposition into control and metacontrol problem.

Algorithm 1 gives the pseudo-code of Bulitko's algorithm when operating under this two-part policy we have outlined. The initial state must be on level 0. At every time step t the agent performs two actions. First, the agent observes the current state s_t, performs an action, a'_t, selected by its metacontrol policy, and the environment transitions to a new state s' with probability $P(s'|s_t, a'_t)$. In the second step, now observing state s', the agent performs an action a_t, selected by the ground policy and transitions to a state s_{t+1} with probability $P(s_{t+1}|s', a_t)$. The process ends when the agent either completes the task by reaching the goal $s_g \in L_N$ or dies (transitions to the death state s_d).

Algorithm 1 Agent Operation

Input: A stochastic decision-making process $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{S}^\dagger, s_0 \rangle$, ground policy $\widetilde{\pi}$, and metacontrol policy π
Output: trajectory $(s_0, s_1, \ldots, s_T), s_T \in \mathcal{S}^\dagger$

```
t ← 0
s_t ← s_0
while s_t ∉ S† do
    a'_t ← π(s_t)                                  ▷ Metacontrol action
    s' ←P(s'|s_t,a'_t)— s_t                        ▷ State transition
    a_t ← π̃(s')                                    ▷ Control action
    s_{t+1} ←P(s_{t+1}|s',a_t)— s'                 ▷ State transition
    t ← t + 1
```

For a fixed ground policy, $\widetilde{\pi}$, we think of the metacontrol as a stochastic control process $\langle \mathcal{S}, A, P, \mathcal{S}^\dagger, s_0 \rangle$ where $P(s'|s, a) = \sum_{\tilde{s}} \mathcal{P}(s'|\tilde{s}, a)\mathcal{P}(\tilde{s}|s, \widetilde{\pi}(s))$. In this work we focus on this *metacontrol process* and the problem of finding the best metacontrol policy given a fixed ground policy.

2.1.1 Measures of Performance

The objective in this level-based environment is twofold: reaching the final level as quickly and as reliably as possible. We measure the quality of a metacontrol policy, π, by: the expected time that agents controlled by π take to reach the goal, s_g; and their *failure rate* or the probability of failing to reach the goal. We can say that one metacontrol policy *dominates* another if it is better on both measures: faster and less likely to result in the agent's death before reaching the goal. However, these measures scores have different scales and trading off one for the other may be application-specific.

Reinforcement learning offers a way to introduce a trade-off between these two measures: at each time step $t = 1, 2, ...$ we assume the agent receives a real-valued reward, r_t, for taking action a_t in state s_t. We say the optimal metacontrol policy is one that maximizes the value function, V^π, defined for any state as

$$V^\pi(s) = E_\pi \left[\sum_{t'=t}^{T-1} r_{t'} | s_t = s \right].$$

Here T is the time point when the agent enters a terminal state $s_T \in \mathcal{S}^\dagger$. The function, $V^\pi(s)$, defines for any state s the expected lifetime return starting from s and following policy π. Then it is possible to measure the performance of different policies by sampling their lifetime return: $V^\pi(s_0)$.

We will make the following assumptions about the rewards in the environments: if two policies are equally reliable the one with lower expected time to reach the goal will have higher expected return, and conversely if two policies are equally fast the more reliable one will have higher expected return. With these assumptions we know that if one metacontrol dominates another then it will also have a higher lifetime return.

Chapter 3

Related Work

In this chapter we explore existing methods of metacontrol that can be applied to the level-based environment.

3.1 Markov Metacontrol Processes

One way to approach the problem is to assume the metacontrol process is a Markov Decision Process (MDP). An MDP is a stochastic decision-making process where on every time step, t, the agent receives a scalar reward r_t with expected value $R(s_t, a_t)$ and the transition probabilities satisfy the *Markov property*: The next state depends on the current state and the action taken but is conditionally independent of all previous states and actions, $P(s'|s_0, a_0, ..., s_t, a_t) = P(s'|s_t, a_t)$.

As in Section 2.1.1 we say that the optimal metacontrol policy, π^*, is one that maximizes the value function, V^π, over all metacontrol policies, π. For an MDP the value function V^π can be expressed recursively using the Bellman equation:

$$V^\pi(s_t) = \sum_{s'} P(s'|s_t, \pi(s_t)) \left(R(s_t, \pi(s_t)) + V^\pi(s')\right). \tag{3.1}$$

This equation has a unique solution [3]: the optimal value function, V^*, that satisfies the Bellman optimality equation:

$$V^*(s_t) \quad = \quad \max_{a \in A} \left[R(s_t, a) + \sum_{s'} P(s'|s_t, a) V^*(s_{t+1}). \right] \tag{3.2}$$

Stochastic Shortest Path MDPs

A metacontrol such as the one described in the previous chapter is looking for the shortest path for the ground policy through the levels of the environment. Assuming

there exists a *proper policy*, one that reaches a goal state from any state s with probability 1, this is a stochastic shortest path problem (SSP) [4].

Value iteration [3] forms the basis for most SSP algorithms. Value iteration starts by initializing state values with an arbitrary value function, V_0, then executes full sweeps of the state space updating every state using the Bellman optimality equation (Equation 3.2):

$$V_n(s_t) = \max_{a \in A} \sum_{s_{t+1}} P(s_{t+1}|s_t, a) \left(R(s_t, a) + V_{n-1}(s_{t+1})\right) \tag{3.3}$$

where $n = 1, 2, ...$ is the iteration number. This update is called a *Bellman update*. Because value iteration includes a full sweep of the state space it can be slow and lack memory for solving even small SSPs.

This strategy is impractical in practice since the state spaces are usually large. However, under the assumptions that one is interested only in a policy originating in a known start state, s_0, more efficient algorithms exist. The Find-and-Revise algorithms [5] must start with an admissible value function V_0, (i.e., one that satisfies $V_0 \geq V^*$ for all $s \in S$). At each iteration, a Find-and-Revise algorithm builds the greedy graph of the current value function: the graph of all states reachable from s_0 using some policy π greedy with respect to the current value function. It then searches this graph for a state that is inconsistent with the Bellman optimality equation and updates the value of that state with a Bellman update.

Although the SSP model is fairly general, for some metacontrol problems there is no proper policy. The existence of a death state, s_d, such as in our environment, means no policy can guarantee reaching the goal from all states. Then the problem includes maximizing the probability of reaching the goal. Kolobov et al. [24] have developed an extension of the Find-and-Revise framework to handle more general SSPs that do not have a proper policy. Algorithms in the new framework add to each iteration a step for eliminating *traps*. Traps are strongly connected component of the greedy graph, G^V, with no outgoing edges and no goal states. In short, they are states from which no policy, greedy with respect to the current value function, can reach the goal. Each trap is eliminated by decreasing the value of all states in the trap in such a way that the value function is still admissible.

The drawback of using the algorithms presented above for metacontrol processes is that they require an estimate of transition probabilities in the metacontrol environment. This may not always exist especially since transitions in the

metacontrol process are dependent both on what state will result when a given action is taken (a transition probability distribution) and on what action the ground policy selects (an intra-level action probability distribution).

Reinforcement Learning

Reinforcement learning uses value iteration to find the optimal value function for an MDP. Suppose that at time t the agent has a policy π and an estimate V_t of the state-value function based for this policy. The agent executes its next action, a_t, chosen by the policy π. This induces in the environment a state transition $s_t \to s_{t+1}$ with associated reward r_t. If the estimate, V_t does not accurately represent V^*, the two sides of the Bellman optimality equation may not be equal and the difference is called the *temporal difference error*:

$$\delta_t^{\mathrm{TD}} = r_t + V_t(s_{t+1}) - V_t(s_t). \tag{3.4}$$

Most temporal difference learning algorithms are based on continually updating the value function estimate based on the error found at each step:

$$V_{t+1}(s_t) \leftarrow V_t(s_t) + \alpha \delta_t^{\mathrm{TD}} \tag{3.5}$$

where $\alpha \in [0, 1]$ is the step-size parameter.

Reinforcement learning has been used for metacontrol in MDPs in Hierarchical Reinforcement Learning (HRL) [12, 17, 22, 40, 34, 43]. The goal of HRL is to combat the high dimensionality of the state space in complex environments by breaking the process down into *activities*: policies on a subset of the state space. This requires a generalization of the MDP called the semi-Markov decision process [20]. In semi-MDPs there is no longer a single time step between actions instead the system is treated as remaining in each state for a random waiting time before transitioning to the next state.

In the option framework of Sutton, Precup, and Singh [44] each activity is defined as an *option* composed of a policy and a termination condition. At each point the agent selects an option and operation proceeds according to the policy of that option until it terminates and a new option is selected. The option policies are generally provided or learned *a priori* by treating each activity as a separate reinforcement learning problem.

Similarly, Diettrich's MAXQ method [13] describes each activity as a separate semi-MDP by defining for each one a local reward function. These local reward

functions are only used during learning to develop a locally optimal policy for each activity. The value function of the initial problem decomposes into a sum of value functions for each activity which form the basis of the learning algorithm. The algorithm learns the hierarchical policy by jointly learning a locally optimal policy for each activity. However, this method rests on the ability to construct good local reward functions. If the local reward function does not accurately reflect global goals the resulting hierarchical policy may be sub-optimal because the optimal policy requires a locally sub-optimal policy for some activity.

The Hierarchy of Abstract Machines method of Parr and Russell [33] defines each activity as a stochastic finite-state machine where certain state transitions can cause actions to be taken in the environment. Each machine has specified *choice states*: non-deterministic states where actions are determined by a learned policy. This allows the programmer to partially specify the activity and constrain the permitted actions without giving a full definition of the policy.

One of the drawbacks of reinforcement learning is that it requires the design of a reward signal. A single scalar must impart to the agent enough information for it to learn to predict how the environment will respond to its actions. In a complex environment it may be difficult to design a reward function that can express and balance all goals and desired behaviours.

On extended sequential decision making processes the rewards may also be sparse or delayed. Consider, for example, an agent doing path-finding on a 1000×1000 map. If the goal state is a single cell on the map and the agent receives a reward of -1 for each step until it reached the goal. It may take a long time for random exploration to even discover the target and to learn which of the moves a long the way were correct requires averaging over many visits.

Another drawback of both reinforcement learning and SSP is that to guarantee convergence to an optimal policy requires that the environment of the metacontrol be an MDP (or at least a semi-MDP). This will not be the case, for example, when information relevant to identifying the state of the environment is missing from immediate observation (partially observable MDPs). Partially observable MDPs are often computationally intractable to solve exactly and performance of reinforcement learning algorithms based on temporal difference learning can be unreliable in this setting [48]. Other algorithms have been developed [26, 19, 36, 37] but they will yield only an approximate solution.

3.2 Metacontrol through Intrinsic Rewards

Researchers have also experimented with using intrinsic rewards for metacontrol. This may be viewed as a metacontrol that, rather than having access to inter-level actions as we define them, augments the reward stream of the environment with additional rewards that we call *intrinsic* rewards. For example, in a level-based environment the metacontrol might encourage an agent to explore a particular level by giving high intrinsic rewards within that level. As such they are not directly comparable to an inter-level action-based metacontrol but they do, nonetheless, serve a similar role of monitoring and guiding a ground policy.

The concept of *curiosity* in AI agents, later to become the formal theory of creativity, fun, and intrinsic motivation, was first suggested by Schmidhuber [39]. It is based on the concept of rewarding actions that improve the agents model of the environment. In this approach the agent has a *model*: a predictor trained to predict future states of environment from previously observed state-action pairs (i.e., an estimate of the transition probabilities of the environment). The metacontrol then generates *curiosity rewards* based on a measure of the mismatch between predictions of the model and reality. In our level-based environment this type of metacontrol encourages the agent to explore levels where the model makes bad predictions, which in turn is expected to improve the model.

Initially the curiosity rewards were directly proportional to the predictor error [39]. This would the agent to levels where prediction error is high and the model needs improvement. A drawback of this approach is that the agent will also have incentive to seek out levels where the environment is less predictable (i.e., the predictor error is high even with the best model).

In his follow-up work [38] Schmidhuber instead based rewards on the *change* in the predictor error with time. Then there is no incentive to seek out levels where the predictor error remains statically high. However, levels where the predictor is improving (reduction in predictor error) are intrinsically rewarding.

Later work by Storck et al. [41] similarly introduced rewards based on information gain: the difference between the model's estimated transition probability distribution before and after a state transition is observed.

Bulitko and Brown [7] introduced similar intrinsic rewards to reinforcement learning agents through a mathematical model of flow. In their framework, flow

was an intrinsic reward, much like Schmidhuber's curiosity reward. However, their agents kept two separate value functions for expected external return and flow return and attempted to maximize a linear combination of the two. They defined for every state $s \in S$ a scalar representing the environmental complexity, $c(s)$, whose range matched an innate ability, a, of each agent. The flow reward of each state $s \in S$ they then defined as

$$f(s) = \frac{1}{|c(s) - a| + \epsilon}$$

where $\epsilon > 0$ is a real-valued constant that ensures that flow rewards are finite.

A potential drawback of using intrinsic rewards for metacontrol is that it is tied to a reward-based ground policy, such as a reinforcement learning algorithm, and thus requires such a ground policy to be applicable to the environment. This may not always be the case as discussed in the previous section.

Another drawback is that if intrinsic and environment rewards are not balanced the agent may be less concerned with actually solving the problem (e.g., collecting the "real" rewards from the environment) than it is with collecting these intrinsic rewards which may, consequently, reduce its lifetime return.

3.3 Dynamic Difficulty Adjustment

Our approach is similar to methods employed for dynamic difficulty adjustment: the process of automatically changing aspects of a video game, as it is played, to avoid the player becoming bored or frustrated with the game (i.e., to keep them in flow). Dynamic difficulty adjustment algorithms observe the behaviour of the player and modify the game content or dynamics to improve the experience. In a sense it is a metacontrol where the agent is a human player. In this case, since the metacontrol has no influence over the agent (the player), it can only control the agent-environment interaction by modifying the environment (the game).

There are two general approaches to dynamic difficulty adjustment: one is to model expected player skills based on the player's behaviour and adjust difficulty to match and the other is to first learn to predict experiences of fun, challenge, or frustration from player input and then use the predictor to direct content.

Numerous approaches exist that modify game complexity based on in-game behaviour. Hunicke and Chapman [21] used probabilistic methods to predict inventory shortfalls (e.g., lack of specific weapons or ammunition) by observing

trends in damage taken and inventory expenditure in a first-person shooter game. When shortfall is predicted a handcrafted policy prescribes adjustment actions based on the health of the player and his inventory. The Interactive Storytelling Architecture for Training [27] models players using a vector of competence levels for different skills. The approach individualizes training by finding the best match between characteristics of available training scenarios and the current state of the skill vector. Lankveld et al. [25] adjust difficulty in a role-playing game based on *incongruity*. Incongruity is defined as the difference between the game complexity and the complexity of the player's mental model of the game, and estimated using the player's health over progress made. The game then controls difficulty by modifying the set of presented enemies to enforce a given level of incongruity.

Approaches based on in-game behaviour lead to platform-specific solutions with explicit rules for how to adjust content to match predicted skills. Other approaches enable more general models by learning to predict directly how content influences player experience. Pedersen et al. [35] train a neural network to predict player self-reported experiences of fun, challenge, and frustration based on measures of player behaviour and in-game content. Yannakakis, Lund, and Hallam [49] use evolving neural networks to predict player self-reported interest based on using certain game features. Zook and Riedl [50] use tensor factorization to correlate time-varying measures of performance in adventure role-playing games to player-reported difficulty. This enables forecasts of players' skill mastery not just in the current state but forward in time. One drawback of using player experience is that it requires data from experiments with human subjects which can be difficult and expensive to collect.

Chapter 4

Flow-seeking Metacontrol

In this chapter we propose a method of using flow to define a metacontrol policy for AI agents in a level-based environment. As described in the introduction, flow in humans appears to emerge only when there is a match between skills and the complexity of the task at hand. By defining a *degree of flow* as a measure of the match between the skills of an agent and the level complexity, we propose a simple metacontrol policy that seeks a state of flow. The policy uses this measure to compare states and evaluate which one is closest to the state of flow.

In this chapter we first define skills, complexity, and degree of flow and then present our algorithm along with an illustrative example.

4.1 Agent's Skills

The goal of our metacontrol is to steer the agent to the right level for its abilities. As such, the metacontrol must observe certain properties of both the agent and the environment to determine the next action. These properties we will now refer to as the *skills* of the agent.

We assume that there exists a representation, $\bar{\sigma} : \mathcal{S} \to \mathbb{R}^d$, of any state as a finite set of skills, quantified by real numbers, so that we can represent the agent's skills in any state s as a d-component vector $\bar{\sigma}(s) \in \mathbb{R}^d$. For convenience we will use the notation $\bar{\sigma}_t = \bar{\sigma}(s_t)$ for the skills of an agent at time t.

For example, in *Angband* the agent's skill may be represented by the character's armour class (AC) and hit points (HP). Then the current skills of the agent in Figure 4.1 are $\bar{\sigma}_t = (10, 20)$.

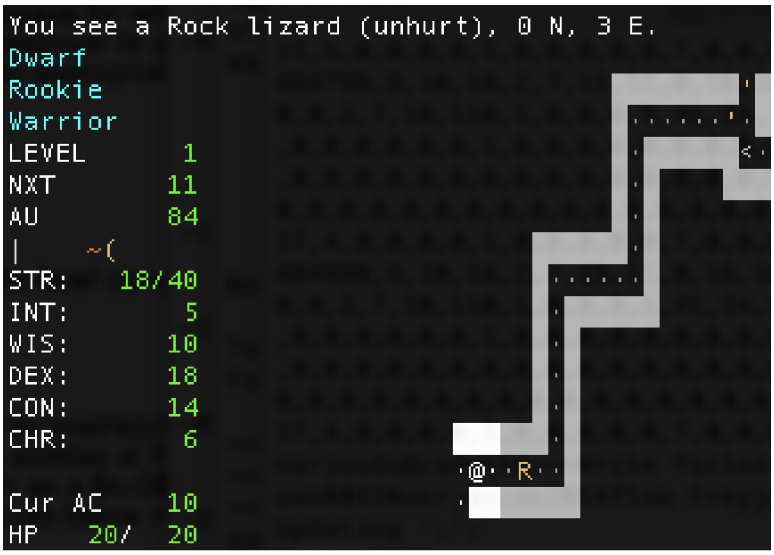

Figure 4.1: The primary statistics in the game of *Angband*.

4.2 Level Complexity

We now consider the complexity of a level for agents with a given ground policy. We say that the *complexity* of a level is the vector of minimum skill values $\bar{c}$ that an agent, using a given ground policy, should possess in order to survive that level and achieve the goal state.

Let us assume that the skills of an agent on level l are given by a vector $\bar{\sigma}_l \in \mathbb{R}^d$. For each level l and each component k of the skill vector, we consider the probability

$$p_{k,l}(x) = \Pr\{\text{agent has } \sigma_{l,k} < x \,|\, \text{agent survived to goal}\}$$

Then ideally, the kth component of the complexity is the minimum value x such that $p_{k,l}(x) = \Pr\{\text{agent has } \sigma_{l,k} < x \,|\, \text{agent survived to goal}\} = 1$. That is

$$c_k(l) = \inf\{x \in \mathbb{R} \mid p_{l,k}(x) = 1\} \tag{4.1}$$

where the infimum (inf) of a set of real numbers is the largest real number that does not exceed any number in the set. For finite sets, infimum coincides with minimum.

However, in some environments death may be unavoidable to some degree. Thus, more generally we define a tolerance *threshold* $0 \leq \rho \leq 1$, such that the kth component of the complexity vector of level l is defined for a given threshold ρ as

$$c_k^\rho(l) = \inf\{x \in \mathbb{R} \mid p_{l,k}(x) \geq \rho\} \tag{4.2}$$

The *complexity profile* of a level-based environment for a threshold ρ is the mapping $\bar{c}^{\rho} : \{0, ..., N\} \to \mathbb{R}^d$ giving the complexity of each level.

Here we assume that the skills of agents are defined in such a way that a higher value of each skill corresponds to a greater probability of reaching the goal. We further assume that this probability is independent of the value of other skills (see discussion in Section 6.2).

For example, one of the agent's skills in *Angband* is armour class. Let us assume that the probability of reaching the goal decays exponentially with armour class on level $l = 1$, with rate parameter $\lambda = 1/100$. Then the probability of reaching the goal with armour class less than some value σ is given by $p_{\mathrm{AC},1}(\sigma) = 1 - e^{-\sigma/100}$. Using a threshold of $\rho = 5\%$, we find the armour class complexity of level 1 to be $c^{5\%}_{\mathrm{AC}}(1) = -\ln(1-\rho)/\lambda \simeq 5$.

4.2.1 Mining Level Complexity

One way to determine the complexity is to use a population of *probe agents* (with some metacontrol policy, π, and ground policy, $\widetilde{\pi}$) and estimate the probability used to define the complexity from sample proportions in these probe agents.

For a given population of n probe agents, let $\bar{\sigma}^i_l$ be the skills of agent i upon entering level l. Let us also define $\hat{p}^i_{k,l}$ to be the fraction of probe agents with higher skill at level l that eventually reached the goal. Then for each k we estimate the complexity of level l for threshold ρ as

$$\hat{c}^{\rho}_k(l) = \min_i \left\{ \sigma^i_{k,l} : \hat{p}^i_{k,l} \geq \rho \right\}. \tag{4.3}$$

Algorithm 2 provides pseudo-code for this method.

Consider an agent in *Angband* with the two skills of rumour class and hit points as before. The goal state is on level $l = 1$ and there are two possible inter-level actions possible: to stay or to descend. We would like to estimate the complexity with a threshold of $\rho = 5\%$, using $n = 3$ probe agents.

The first probe agent trial, $i = 1$, begins on line 5 and lines 6-9 initialize parameters for the trial. The probe agent begins operation on line 10, with the metacontrol choosing to stay (line 14) until time $t = 386$ when the metacontrol chooses to descend. In the next iteration of the loop (line 10) the condition on line 11 becomes true ($l_{387} = 1 > 0$) and on line 12 $\bar{\sigma}^1_1$ is set to $\sigma_{387} = (10, 17)$ then l gets

Algorithm 2 Complexity Mining

Input: A metacontrol process $\langle \mathcal{S}, A, P, \mathcal{S}^\dagger, s_0 \rangle$ with N levels,
1: a probe agent policy π,
2: a skill representation $\bar{\sigma}$,
3: a threshold ρ,
4: and a number of probe agents n.
Output: complexity profile: $\bar{c}(l)$ for all levels $1 \leq l \leq N$.
5: **for** $i = 1, 2, ..., n$ **do**
6: $e_i \leftarrow 0$ ▷ $e_i = 1$ if probe i was successful in reaching the goal, 0 otherwise
7: $t \leftarrow 0$
8: $s_t \leftarrow s_0$
9: $l \leftarrow 0$ ▷ l is the maximum level recorded for probe i
10: **while** $s_t \notin S^\dagger$ **do**
11: **if** $l_t > l$ **then** ▷ If this is the first time probe i visits level l
12: $\bar{\sigma}_l^i \leftarrow \bar{\sigma}(s_t)$ ▷ $\bar{\sigma}_l^i$ are the skills of probe i upon entry to l.
13: $l \leftarrow l + 1$ ▷ Maximum level increased
14: $a_t \leftarrow \pi(s_t)$ ▷ Metacontrol acts
15: $s_{t+1} \xleftarrow{P(s_{t+1}|s_t, a_t)} s_t$ ▷ State transition
16: $t \leftarrow t + 1$
17: **if** $s_t = s_g$ **then**
18: $e_i \leftarrow 1$ ▷ Probe i was successful
19: **for all** i, k, l **do** ▷ Find the fraction of successful probes with less skill on level l.
20: $p_{k,l}^i \leftarrow \frac{\#\{j:\, \sigma_{k,l}^j \leq \sigma_{k,l}^i \text{ and } e_j = 1\}}{\#\{j:\, e_j = 1\}}$
21: **for all** k, l **do** ▷ Find the minimum skills with fraction $p_{k,l}^i$ over the threshold.
22: $c_k(l) \leftarrow \min_i \left\{ \sigma_{k,l}^i :\, p_{k,l}^i > \rho \right\}$

incremented and agent operation continues until $t = 410$ when they reach the goal s_g. Then the condition on line 17 is true so $e_1 = 1$.

This for-loop (line 5) continues for $n = 3$ probe agents, with probe agent 2 surviving to the goal ($e_2 = 1$) with skills $\sigma_1^2 = (8, 20)$ on level 1 and probe agent 3 failing to reach the goal ($e_3 = 0$) with $\sigma_1^3 = (5, 16)$.

Then for $n = 1, 2, 3$ probe agents, $k = 1, 2$ skills and level $l = 1$ line 20 finds the fraction of successful probes with skill k less than probe i on level l: $p_{1,1}^1 = 2/2 = 1$, $p_{2,1}^1 = 1/2$, $p_{1,1}^2 = 1/2$, $p_{2,1}^2 = 2/2 = 1$, and $p_{1,1}^3 = p_{2,1}^3 = 0$. Finally, for $k = 1, 2$ and $l = 1$ in line 22 we find the complexity of each skill: $c_1(1) = \min\{8, 10\} = 8$ and $c_2(2) = \min\{17, 20\} = 17$.

We think of these probe agents as testing the limits of what an agent can accomplish with a given skill. The accuracy of the estimate (i.e., how closely the estimated complexity in Equation 4.3 matches the actual complexity given by Equation 4.2) depends on how the collected data of the probe agents is distributed in the space of possible skills. For example, cautious probe agents (that attempt

to maintain overqualification to avoid failure) may yield deceptively high estimates of the minimum: on some dimensions the actual complexity may be lower than the probe agents revealed simply because none of the probe agents dared proceed with less skill.

The accuracy of the estimate also depends on how well the probe agent sample represent the population of agents for which the complexity is mined. If the probe agent policy is different from the flow-seeking agents, the probability of reaching the goal with a given skill may also be significantly different thus making the mined complexity inaccurate.

4.3 Degree of Flow

We are now ready to define, for a given agent, the degree of flow the agent is experiencing. The model we use in this work is an extension of previous work by Bulitko and Brown [7] and Bulitko [6]. Formally, the degree of flow experienced by an agent is at time t on level l is given by

$$F(\bar{\sigma}_t, \bar{c}(l)) = \frac{1}{||\bar{\sigma}(s_t) - \bar{c}(l)|| + \epsilon}$$

where $\epsilon > 0$ is a real-valued constant that bounds the flow and $||\cdot||$ is the Euclidean distance: $||\bar{p} - \bar{q}|| = \sqrt{\sum_{i=1}^{d}(p_i - q_i)^2}$ for all $p, q \in \mathbb{R}^d$. The degree of flow, F, takes on a maximum of $1/\epsilon$ when the skills of the agent, $\bar{\sigma}_t$, and the complexity of the level, $\bar{c}(l)$ are equal (i.e., in the state closest to flow).

4.4 A Flow-Seeking Metacontrol Policy

Our flow-seeking metacontrol policy proceeds in two stages. In the first stage, using a basic form of metacontrol, we mine the level complexity for each level, $\bar{c}(l)$ as described in Section 4.2.1. From this we can construct a flow-seeking metacontrol policy that operates according to Algorithm 3.

At each time point, t, the agent observes the skill representation of the current state $\bar{\sigma}_t$ and takes the inter-level action that maximizes the degree of flow experienced on the intended level

$$a = \underset{a' \in A}{\operatorname{argmax}} F\left(\bar{\sigma}_t, \bar{c}(l_{a'})\right).$$

Here $l_{a'}$ is the intended level of the inter-level action, a'. We will use $F(a') = F(\bar{\sigma}_t, \bar{c}(l_{a'}))$ as short-hand notation for the decree of flow expected on the intended level of inter-level action a'.

For example, consider an agent in *Angband* with two skills: armour class and hit points, $\bar{\sigma}_t = (10, 20)$. The agent is on level 1 and has two available inter-level actions: stay on level 1 (a_1) or go to level 2 (a_2). The complexity of level 1 is $\bar{c}(1) = (5, 20)$ while the complexity of level 2 is $\bar{c}(2) = (8, 24)$.

Then if $\epsilon = 1$ for a_1 we have $F(a_1) = \frac{1}{||(10,20)-(5,20)||+1} = 1/6$ while for a_2 we have $F(a_2) = \frac{1}{||(8,24)-(5,20)||+1} = 1/4$. So the metacontrol will select a_2 and go to level 2.

Algorithm 3 Flow-seeking metacontrol

Input: A metacontrol process $\langle \mathcal{S}, A, P, \mathcal{S}^\dagger, s_0 \rangle$ with N levels,
a skill representation $\bar{\sigma}$,
a complexity profile $\bar{c}(l)$ for all levels $1 \leq l \leq N$,
and a constant $\epsilon > 0$.

$t \leftarrow 0$
$s_t \leftarrow s_0$
while $s_t \notin S^\dagger$ **do** ▷ While state is not terminal
  for all $a' \in A$ **do**
    $F(a') \leftarrow \frac{1}{||\bar{\sigma}_t - \bar{c}(l_{a'})||+\epsilon}$ ▷ Compute degree of flow for each inter-level action
  $a_t \leftarrow \text{argmax}_{a' \in A} F(a')$ ▷ Take the action with the highest degree of flow
  $s_{t+1} \xleftarrow{P(s_{t+1}|s_t, a_t)} s_t$ ▷ State transition for metacontrol (includes the action of the ground policy)
  $t \leftarrow t + 1$

4.5 An Illustrative Example

Bulitko [6] devised a simple environment to test the method we have described. Here we will use his environment to help visualize flow-seeking as a metacontrol and discuss our conclusions about the results in this environment.

4.5.1 The Environment

The environment was designed to focus only on metacontrol. It has $N + 1$ disjoint levels where each level, $0 \leq l \leq N$, has only one state, s^l (i.e., $L_l = \{s^l\}$ for all $0 \leq l \leq N$). The initial level s^0 is our start state and the final level is our goal state, $s_g = s^N$. Thus, the state space is composed of $N + 1$ states and the death state, $\mathcal{S} = \bigcup_l L_l \cup s_d$. Each state has $N + 1$ inter-level actions a that take the agent

to level l_a. Figure 4.2 depicts such an environment with 3 levels. For simplicity, the death state is not depicted but every action leads to the death state, s_d, with a certain probability.

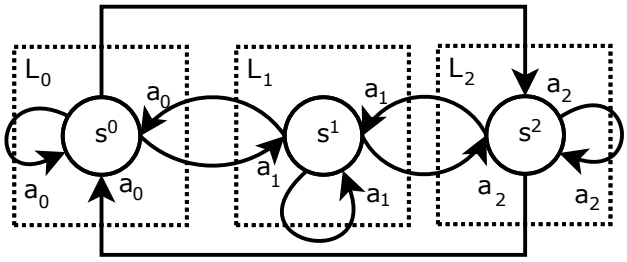

Figure 4.2: State diagram for a 3-level environment. All actions can also lead to the death state (not depicted).

In this environment, an agent's skill is defined by their *age*, $\sigma_t = t$, the number of turns it has lived. For each level, $0 \leq l \leq N$, there is a minimum age requirement (or complexity), $c(l) \in \mathbb{R}$. On each turn, t, the probability of dying is proportional to the deviation between its age, t, and the age requirement, $c(l_t)$:

$$\forall a \in A : \mathcal{P}(s_d | s^{l_t}, a) = \begin{cases} p^\dagger & \text{if } c(l) \leq t \\ \min\{1, p^\dagger + \tanh\left(c(l_t) - t\right)\} & \text{if } c(l) > t \end{cases} \tag{4.4}$$

where $p^\dagger \in [0, 1]$ is the *ambient probability of death* in the environment; regardless of level or age. If skill exceeds the requirement for the current level then ambient death becomes the only cause of death in the environment. However, if an agent is younger than the required age for the current level, the probability of dying increases proportionally to the deviation from the required skill. This is illustrated in Figure 4.3.

At each time step, t, the agent selects an action a_t that causes the environment to transition from state s_t to state s_{t+1} and the agent receives a reward r_t. The reward is given by

$$r_t = \begin{cases} l(s_t) & : s_{t+1} \notin \mathcal{S}^\dagger = \{s_d, s_g\} \\ -\sum_{t'=0}^{t-1} r_{t'} & : s_{t+1} = s_d \\ (T_{\max} - t) \cdot N & : s_{t+1} = s_g \end{cases} \tag{4.5}$$

where $T_{\max}$ is the maximum time given to reach the goal.

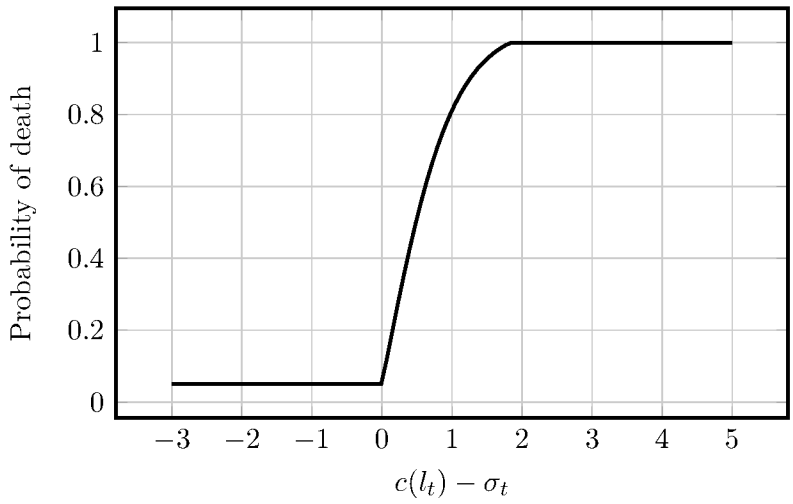

Figure 4.3: Probability of death as a function of deviation from the required skill of an agent when ambient probability of death is $p^\dagger = 5\%$.

The intuition for this reward function is that agents receive a reward equal to the current level throughout their lifetime but forfeit everything if they die before reaching the goal (second line in Equation 4.5). If they make it to the goal they can live to the maximum time collecting the rewards of the highest level, N (third line in Equation 4.5). This ensures that reaching the goal is more desirable than staying on level $N - 1$ (the highest rewarding level) to prolong the trial and receive more rewards.

Figure 4.4 compares the lifetime rewards of two agents both reaching the goal but at different times. Since agent 1 reaches the goal sooner he spends a longer time collecting the reward of the final level and his lifetime rewards are greater.

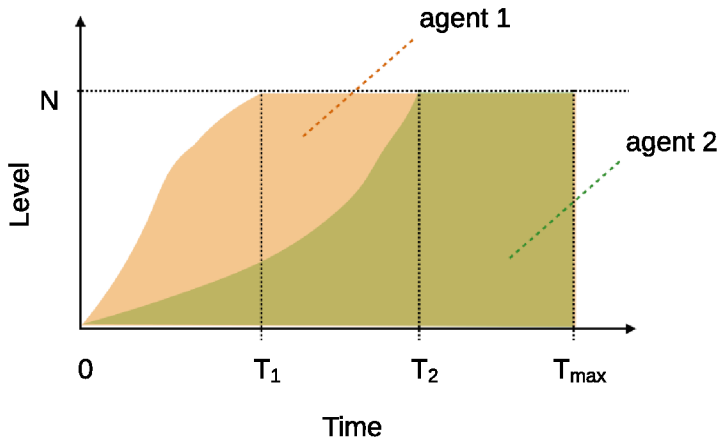

Figure 4.4: Schematic of the level progression of two agents. Agent 1 reaches the goal first so his lifetime return (area under the curve) is greater. Figure is taken from Bulitko [6].

This environment was designed to reward agents for maintaining a match between the agent's skills and the level complexity. Agents are rewarded more for taking on more complex levels but if they are underqualified their probability of dying increases fast.

4.5.2 Empirical Study

Bulitko [6] considered two environments with different complexities: a quadratic complexity, $c(l) = l^2$, $0 \leq l \leq 40$; and a square root complexity $c(l) = \sqrt{l}$, $0 \leq l \leq 200$. In his initial experiment he assumed the levels were continuous. We reproduced Bulitko's study with the modification of assuming discrete levels to adapt the environments to our problem definition.

In these environments he compared two types of metacontrol: a flow-seeking policy and a simple baseline. The *baseline policy* maintained a constant *progression rate*, $\alpha \in \mathbb{R}_+$, through the levels. At each turn, t, of its lifetime, a baseline agent with progression rate α will be on level $l_t = \min\{\lfloor \alpha \cdot t \rfloor, N\}$ (in Bulitko's experiment [6] he allowed agents to be on level $l_t = \alpha \cdot t$ since levels were continuous). For example, if a baseline agent with $\alpha = 0.5$ has survived to turn $t = 11$, it will be on level $l_{11} = \lfloor 0.5 \cdot 11 \rfloor = 5$.

Mining Level Complexity

In this environment, an agent's skill is defined by their age, $\sigma_t = t$. Correspondingly, complexity of each level, l, is the age requirement of that level, $c(l)$. To mine level complexity Bulitko [6] used a method similar to the one described in Section 4.2 whereby the complexity is learned through observation of probe agents.

Each probe agent is given a fixed progression rate $\alpha > 0$ and two flip points, $0 < f_1 < f_2 < N$. All probe agents follow the behaviour of a baseline agent parameterized by some α until they reach the level given by their first flip point, f_1. At that point they will randomly decide to either slow down or speed up. If they speed up, their new progression rate, α', will be uniformly drawn from the range $(0, N - f)$. If they slow down, α' is uniformly drawn from the range $(0, \alpha)$. In either case, they maintain the new progression rate until they reach the second flip point, f_2, where they slow down by choosing a new progression rate uniformly drawn from $(0, \alpha')$.

For each level $0 < l < N$ the estimated complexity was the shortest time taken to reach level l such that the fraction of faster probe agent that survived to the goal

is less than a given threshold ρ. Therefore, for each level the fastest ρ percent of surviving probe agents were removed before taking the minimum.

For each environment we considered 8000 probe agents with initial progression rate α in the range $[0.001, 1]$ and threshold $\rho = 0.1\%$. For the quadratic complexity the ambient death probability was $p^\dagger = 0.01\%$ but for the square root complexity we used $p^\dagger = 0.1\%$. The complexity we mined was close to the actual complexity in both environments (Figure 4.5).

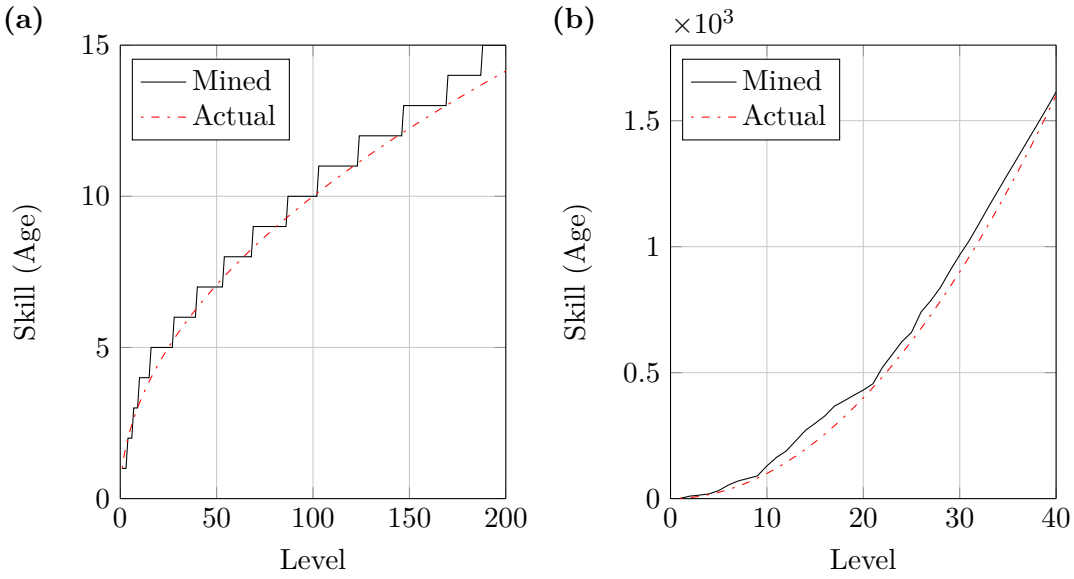

Figure 4.5: Actual and mined complexity for the environment. In (a) the actual complexity is quadratic, $c(l) = l^2$, and in (b) the actual complexity is a square root, $c(l) = \sqrt{l}$.

Experiment using a square root complexity

In this experiment we used a square root complexity, $c(l) = \sqrt{l}$, $0 \leq l \leq 200$, and an ambient probability of dying $p^\dagger = 0.1\%$. Flow-seeking agents were compared to baseline agents with 50 different progression rates, α, equally spaced in the range from 0.5 to 3. A 1000 trials were run for each α and a 1000 trials were also run for the flow agents. The mean return ($\pm$ standard error of the mean) of the flow-seeking metacontrol ($7.72 \times 10^4 \pm 0.33 \times 10^4$) was higher than that of the best baseline policy ($5.04 \times 10^4 \pm 0.76 \times 10^4$) which the had progression rate $\alpha = 1.15$. The flow-seeking metacontrol was also faster and more reliable than the baseline policy for all progression rates considered (Figure 4.6a).

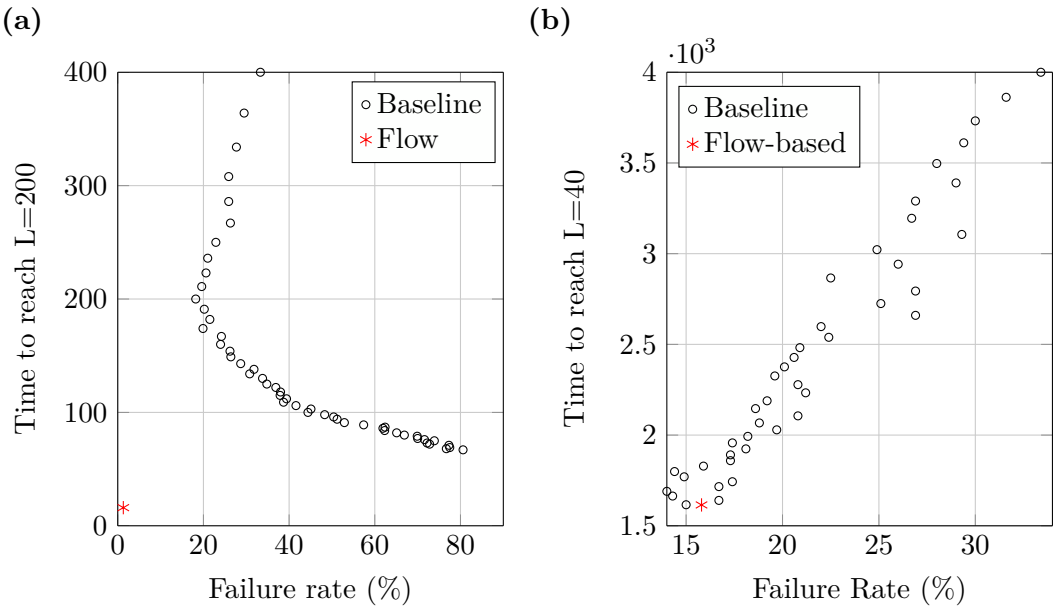

Figure 4.6: Average time to reach the goal and failure rates of different metacontrol policies. In (a) complexity is quadratic and policies considered were flow-seeking and 50 different baseline policies with progression rates equally spaced in the range from 0.01 to 0.028. (b) uses square root complexity and 50 baseline progression rates equally spaced in the range from 0.2 to 2.5.

Experiment using a quadratic complexity

In this experiment we used a quadratic complexity, $c(l) = l^2$, $0 \leq l \leq 40$, and an ambient probability of dying $p^\dagger = 0.01\%$. Again we ran a 1000 trials each of 50 different progression rates, equally spaced in the range from 0.01 to 0.028, and a 1000 trials were also run for the flow agents. The mean return ($\pm$ standard error of the mean) of the flow-seeking metacontrol ($11.75 \times 10^4 \pm 0.15 \times 10^4$) was higher than that of the best baseline policy ($10.86 \times 10^4 \pm 0.14 \times 10^4$) which the had progression rate $\alpha = 0.024$. There was no difference in speed or reliability between the best baseline policy and the flow-seeking policy (Figure 4.6b).

4.5.3 Discussion

This experiment illustrates the application of the flow-seeking metacontrol policy and the type of environment where it performs well. It also highlights a notable difference between the two-dimensional performance measure of speed and reliability and the reward-based performance measure. As Figure 4.6b shows in the quadratic complexity environment there is no difference between the best baseline agent and

the flow-seeking agents in speed or reliability, but there is still a significant difference in lifetime return.

This difference is due to the reward structure favouring agents spending time on higher levels - taking on the highest possibly complexity that they can survive. The flow-seeking agents collect close to the maximum possible return by progressing fast through the lower levels and then slowing down at higher levels. The fixed progression rate of the baseline forces agents to move slower at the lower levels (or they would move too fast through the higher levels). This decreases their return as they spend less time of their time at the high levels.

As an example, Figure 4.7 shows level progression of a single run of the flow-seeking and best baseline policy. The flow-seeking agent spent approximately 500 turns reaching level 20 while it took the baseline agent approximately 800 turns. The difference of approximately 300 turns the flow-seeking agent spends on the last 20 levels instead, which translates into higher return.

A reward structure where reward increases with challenge is common in level-based environments such as video games. Under those circumstances seeking flow may increase the return as the example above illustrates.

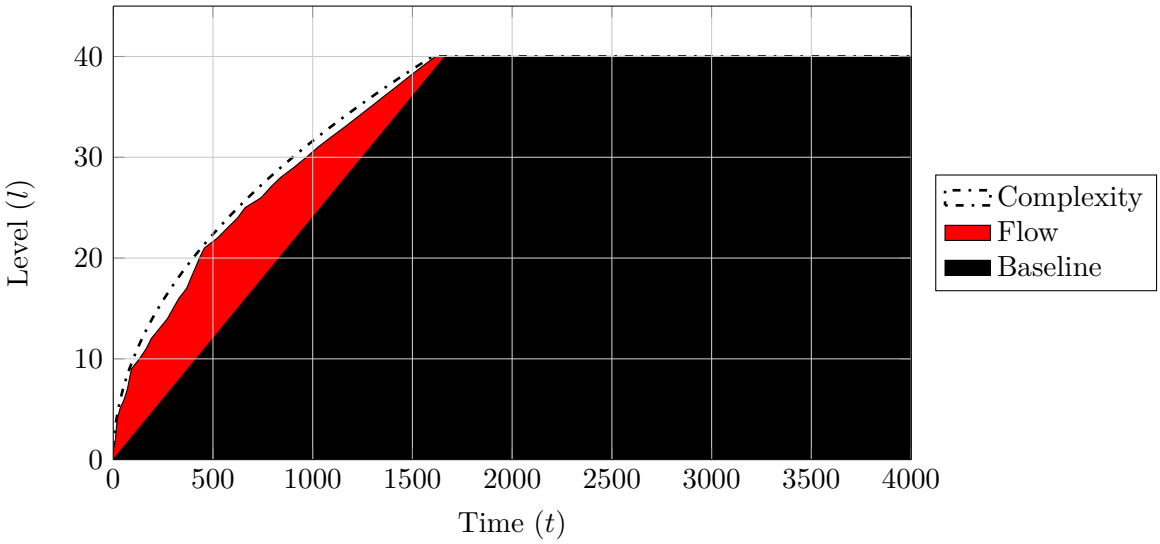

Figure 4.7: Level progression of a typical flow-seeking and baseline agent. The flow-seeking agent achieves a higher lifetime return (area under the curve) despite reaching the goal at the same time as the baseline agent ($t = 1600$).

Chapter 5

Empirical Study in *Angband*

This chapter provides an empirical evaluation of the flow-seeking algorithm in the *Angband* role-playing game. Section 5.1 describes the *Angband* test bed and the existing automatic player used for the ground policy and as a baseline for the metacontrol. Our methods for defining skills and mining the level complexity in *Angband* are described in Section 5.2. Section 5.3 describes the results of an experiment comparing flow-seeking metacontrol to the baseline automatic player and to a uniformly random metacontrol policy.

5.1 The *Angband* Environment

The metacontrol problem we consider in our empirical study is that of level progression in the game of *Angband*. *Angband* is an open-source role-playing game originally developed by Cutler and Astrand in 1990 by building on another game, UMoria 5.2.1. Since then many have been involved in its development; most notably by Swiger, Harrison and Ruehlmann who did extensive work maintaining and expanding the code to get it to the current version.

The game begins in town with shops that allow the player to purchase supplies. On the outskirts of the town lies the entrance to the dungeon of *Angband*. The dungeon is a cave composed of multiple levels; each level has approximately 50 rectangular rooms connected by tunnels. The player assumes the character of an adventurer that roams the dungeon searching for gold and fighting the monsters within each level. In the dungeon the character also picks up equipment and magical items such as weapon or scrolls and these help them gain an advantage in combat.

Staircases exist on each level to allow the player to move both up and down within the dungeon. Note that in *Angband* dungeon levels are numbered from shallowest

to deepest: the town is 0 and the deepest dungeon level is 100. Thus moving to a shallower dungeon level means going to a lower level number, and conversely moving to a deeper dungeon level means going to a higher level number.

Figure 5.1 shows a partial visualization of the game state in *Angband*. The player is represented by the '@' sign and monsters are generally represented by letters of the alphabet (e.g., the 'o' is a black orc). The '<' above the player is an up staircase allowing the player to move to a new dungeon at a shallower dungeon level.

Figure 5.1: A partial visualization of the game state in *Angband*.

5.1.1 The Character and Agent Skills

The player controls a character exploring in the dungeon of *Angband*. This character has three primary attributes chosen by the player: sex, race, and class. The sex is purely for flavour, but the race and class have an effect on fighting ability. There are eleven different races and six classes.

Each character has six primary *statistics*: strength (STR), intelligence (INT), wisdom (WIS), dexterity (DEX), constitution (CON), and charisma (CHR) in addition to hit points (HP) and armour class (AC). These are displayed on the left-hand side of the screen in Figure 5.1. Characters also have other abilities such as speed, magic devices, stealth, searching ability, fighting skill, and shooting skill. All these abilities can be modified by magic, equipment, race, and class (see the player's guide to *Angband* [1] for a more detailed description). Survival in the

dungeon depends on all of these abilities as well as various supplies and equipment the character carries (e.g., weapons, armour, food, potions, and scrolls).

In Chapter 4 we defined skills, $\bar{a}$, as a mapping from the state space to a d-dimensional real-valued vectors. Each of the d components is a feature of the game and agent state known as a *skill*. There are two types of skills to consider in *Angband*: as the character kills monsters in combat she accumulates experience and abilities but in addition the player is learning new skills (e.g., how to tackle certain monsters, what objects are important).

In this work we assume that we can base metacontrol decisions in *Angband* only on the abilities of the characters. Generally, the skills of the player, not just the character they are playing, may be relevant. For example, for human players the amount of experience of playing the game may be a necessary skill to consider. However, we will consider only agents with a fixed, rule-based ground policy and as such the player skills are static and the only variation in the difficulty the agent faces comes from the abilities of their character.

5.1.2 Scoring

The object of the game is for the character to reach the 100^{th} level and defeat Morgoth, the Lord of Darkness. As the player guides the character deeper into the dungeon, the monsters become more formidable. By our definition, the complexity of level is the minimum skills needed to reach the goal from that level with a certain probability. As the monster become more formidable it requires greater skills for the character to defeat them thus the level complexity is expected to increase the higher the level is (the deeper into the dungeon).

For each kill the character gains experience points and the skills related to character's statistics increase. If the player dies before achieving the goal, scoring is based on the experience points accumulated. We used this score as the reward function for evaluating performance of agents in *Angband*.

The experience points gained per kill are equal to the monster level (a measure of the difficulty of the monster) divided by character level (a measure of the experience points accumulated). Thus, in contrast to the degree of flow, score per kill will increase with increasing difficulty even when there is a mismatch between the skill and the complexity. However, since the likelihood of dying increases the less experienced the character is compared to the monster level, it is our hypothesis that

the expected lifetime experience should be at a maximum for flow-seeking agents - agents that are attempting to match skills and difficulty.

5.1.3 The Borg

The APW Borg (or simply the *borg*) is an open-source automatic player for *Angband* battle simulator developed by White [47]. It uses rule-based logic to make intelligent decisions about inventory and equipment management, combat, level progression and exploration, and other aspects of *Angband*.

The principle of the Borg was to play the game as if it were human (i.e., it reads data displayed on screen and acts on that observation; treating the game as a black box). As a result, the operation is slow and play-outs can only be performed starting from the beginning of the game. However, it is the best automatic player that exists for *Angband*, capable even of winning the game under certain conditions [46] although not frequently, as evidenced by Figure 5.2 showing the fraction of surviving agents as a function of level from an initial population of 1532.

In our experiments we used the borg as the ground policy. Its level progression module serves as a competing method for metacontrol. To do this we modified the APW Borg to separate the metacontrol from the control and allow replacement of the metacontrol with different policies. In addition, although the logical choice for the goal state is the defeat of Morgoth on level 100 (the actual goal of the game), given the performance of the borg in initial experiments we chose a more conservative goal of reaching level 30. Even then only approximately 10% of borg agents could achieve it.

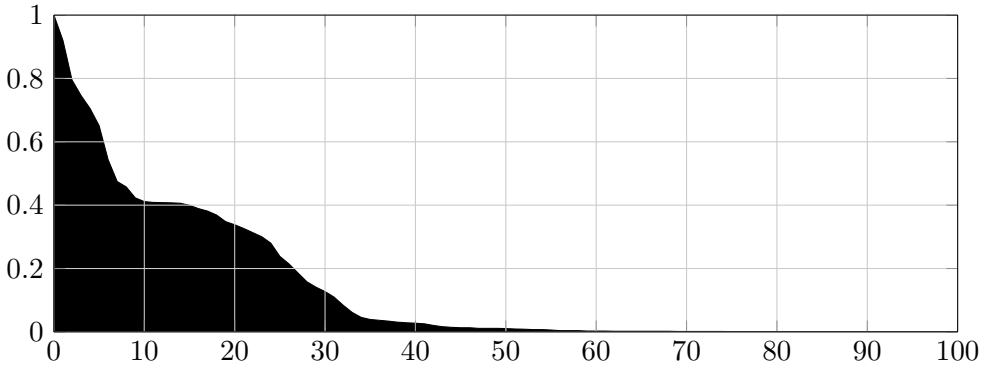

Figure 5.2: Fraction of surviving borg agents as a function of level. Initial population was 1532 but by level 10 less than 50% of them remain and by level 30 the fraction is close to 10%.

5.1.4 Inter-Level Actions

Changing of levels in *Angband* happens for a number of reasons. The player may be forced to leave the level, for example if they fall through a trap door or are magically teleported to another level. The player may also choose to leave to return to town and restock on supplies or to escape from immediate danger. Finally, they may choose to descend because they have completed the current dungeon level and consider themselves prepared for the next level. In this work we considered only the last one to be an inter-level action. The other actions, although they can result in a change of level, are part of ground-level activities (e.g., exploring a dungeon, fighting a monster, restocking supplies) and as such were controlled by the borg.

Choosing to descend to the next level can only be done when the character is standing on a down staircase. However, for a metacontrol to be effective, it requires the ability to take inter-level actions from any state. If it could only descend when the ground policy decides to navigate to a staircase, then it would be the ground policy that is deciding when to descend. Thus, we considered abstracted inter-level actions, which instead of changing levels induce the ground policy to search for a staircase. Essentially, the metacontrol has two available actions: go deeper or stay; but neither will result in an immediate change in the game state. Instead, they result in a change in the ground policy state; giving it the goal of seeking a staircase.

Going to shallower levels was not a part of possible inter-level actions because the borg is implemented to only consider moving to a shallower level as part of the aforementioned ground-level activities. As such, it proved difficult to integrate a metacontrol that goes up with the borg and it might also be considered biasing the experiment towards competing metacontrols.

5.2 Flow-Seeking Metacontrol Initialization

The flow-seeking metacontrol requires a skill representation and corresponding complexity for each level to calculate the degree of flow. In this section we discuss how these were set up for *Angband*.

5.2.1 Skill Selection

In this section we explore different feature selection methods used for determining skills in *Angband*.

The borg, extracts and operates on 202 features that form its complete observation of all the supplies and abilities of the character. However, not all of these features appear equally relevant to a metacontrol policy. In this work we considered two skill sets drawn from these features: the 20 features used by the borg metacontrol and a subset selected using correlation feature selection [15]. Correlation feature selection looks to maximize the mean correlation between each feature and a given value, while minimizing the mean pairwise correlation between features of the subset.

In choosing skills we are seeking features that will give the best metacontrol. Our measures of performance are the expected time and failure rate of the agents. However, we expected our complexity mining scheme would ensure that the flow-seeking metacontrol minimizes expected time. Complexity is estimated as a minimum of skills needed to survive to the goal and the flow-seeking metacontrol will descend as soon as its skills develop to match the complexity of the next level. This means in the absence of complexity the flow-seeking metacontrol is descending as fast as possible. What we desire is a complexity that balances this with our other goal of surviving all the way to the final level. As a result, we chose to look for the features that are correlated with survival to the goal.

We used borg performance data from 3000 runs. The data consisted of all 202 borg features upon entry to each level; classified by survival to the goal. This data was passed to Weka's [16] `CfsSubsetEval` class with default options and yielded a subset of 26 features. The two methods described yielded two different skill sets (Table 5.1) to use with our flow-seeking metacontrol.

5.2.2 Mining Level Complexity in *Angband*

We defined level complexity $\bar{c}(l)$ at level l as the minimum skill below which agents survive to the goal, $L_{\max} = 30$, with low probability. To estimate the complexity of each level we first run probe agents and observe their performance.

In *Angband* these probe agents are regular borg agents with some added randomization. Each probe agent uses the borg metacontrol policy until it crosses a given flip point $0 \leq l' \leq L_{\max}$. After the flip point, the metacontrol will always choose to descend. The pseudo-code of the probe agent metacontrol is given in Algorithm 4.

Table 5.1: Skill sets used by the flow-seeking metacontrol. Values in bold are in both skill sets.

CFS Skill Set	*Borg Skill Set*
Amount of Potions of Healing	Amount of Food
Base Intelligence	Amount of Fuel
Base Wisdom	**Speed**
Base Dexterity	Access to Resistance to Fire
Base Charisma	Access to Resistance to Cold
Base Constitution	Access to Resistance to Electricity
Strength (with Augmentations)	Hit Points
Constitution (with Augmentations)	Character Level
Charisma (with Augmentations)	**Amount of Potions of Healing**
Self-Illuminating	Access to Resistance to Acid
Speed	Access to Resistance to Chaos
Resistance to Acid	Access to Resistance to Disenchantment
Resistance to Poison	Access to Resistance to Drain Life
Resistance to Fear	Access to Resistance to Paralysis
Resistance to Nexus	Amount of Potions of Cure Critical Wounds
Access to Resistance to Cold	Extrasensory Perception
Access to Resistance to Poison	**Self-Illuminating**
Access to Resistance to Fear	Light Radius
Access to Resistance to Light	Amount of Scrolls of Teleportation
Damage per Hit	Amount of Teleportation Effects
Weapon's Damage per Hit	
Weapon's Damage by Cold	
Amount of Scrolls of Phase Door	
Amount of Scrolls Teleport Other	
Amount of Potions of Resistance to Cold	
Amount of Attack Rods	

We ran 833 probe agents. As described in Section 4.2.1, at the end of all the probes we filter out all data from probe agents that did not reach $L_{\max}$. For each level $l \in \{1, \ldots, L_{\max}\}$ and each component k of the skills we then remove the bottom $\rho = 10\%$ and take the per-component minimum.

Algorithm 4 Probe Agent Metacontrol

Input: The borg metacontrol policy π, a state s, and a flip point l'
Output: An inter-level action a

```
if s ∉ S† then
    a ← π(s_t)                          ▷ Borg metacontrol action
    if l(s) > l′ then                   ▷ If current level is higher than flip point
        a ← a_{l(s)+1}                  ▷ Choose to go to next level
```

5.3 Experiment

To investigate if flow-seeking metacontrol improves performance in *Angband* we collected performance data for two different flow-seeking policies, which we call `FlowDesigned` and `FlowAuto`. These flow-seeking policies were compared to the borg metacontrol and a random metacontrol. The details of the four metacontrols are given in Table 5.2.

We ran 750 trials for each of the four metacontrols considered. All four agent types used the same ground policy used by the borg. Each trial consisted of an agent starting from the initial state of the game and proceeding until they died or reached the goal, $L_{\max} = 30$. Controllable initial character statistics were set to be the same for all agents (Table 5.3).

The performance measures used to compare metacontrols consisted of average turns taken to reach the goal, failure rate, and mean score. For all means the measure of dispersion used was the 95% t-based confidence interval.

Comparison of failure rates was done using a chi-squared test with Marascuillo *post-hoc* test [28]. For comparing mean score and average turns among different metacontrols we used a permutation F-test with Holm-Bonferroni corrected *post-hoc* [18] using 5000 permutations chosen at random without repetition. For all statistical tests the significance level was chosen *a priori* to be $\alpha = 5\%$.

Table 5.2: Description of the four metacontrols considered in our experiment.

Name	Description
`FlowDesigned`	Flow-seeking metacontrol with a designed skill set from the features used by the borg.
`FlowAuto`	Flow-seeking metacontrol with an automatically selected skill set using correlation feature selection.
`Borg`	The borg metacontrol.
`Random`	Metacontrol that on each turn picks from the available inter-level actions with equal probability.

Table 5.3: Summary of controllable initial character statistics for all agents. Values of all other abilities are initialized by the game engine.

Sex	Race	Class	STR	INT	WIS	DEX	CON	CHR
Female	Dwarf	Warrior	17	10	10	18	10	10

5.3.1 Results

Metacontrol had a significant effect on both average time to reach the goal (permutation F-test, $p < 0.0002$) and failure rate (chi-squared test, $\chi^2 = 15.15$,$p = 0.0017$) as shown in Figure 5.3.

`FlowDesigned` agents were the fastest to reach the goal followed by `random` and `borg` agents (with no significant difference) and the slowest were the `FlowAuto` (HolmBonferroni *post-hoc* permutation test). `FlowDesigned` and `random` agents had similar failure rates while the failure rates of `borg` and `FlowAuto` agents were lower than `FlowDesigned` by approximately 5% (Marascuillo *post-hoc* test; $p < 0.05$).

Metacontrol also had a significant effect on the score (permutation F-test, $p = 0.012$) as shown in Figure 5.4. The average score of `borg` agents was significantly higher than the average score of `random` and `FlowDesigned` agents. In addition, the average score of the `FlowAuto` agents was also significantly higher than that of `FlowDesigned` agents (Holm-Bonferroni corrected *post-hoc*; $p < 0.05$).

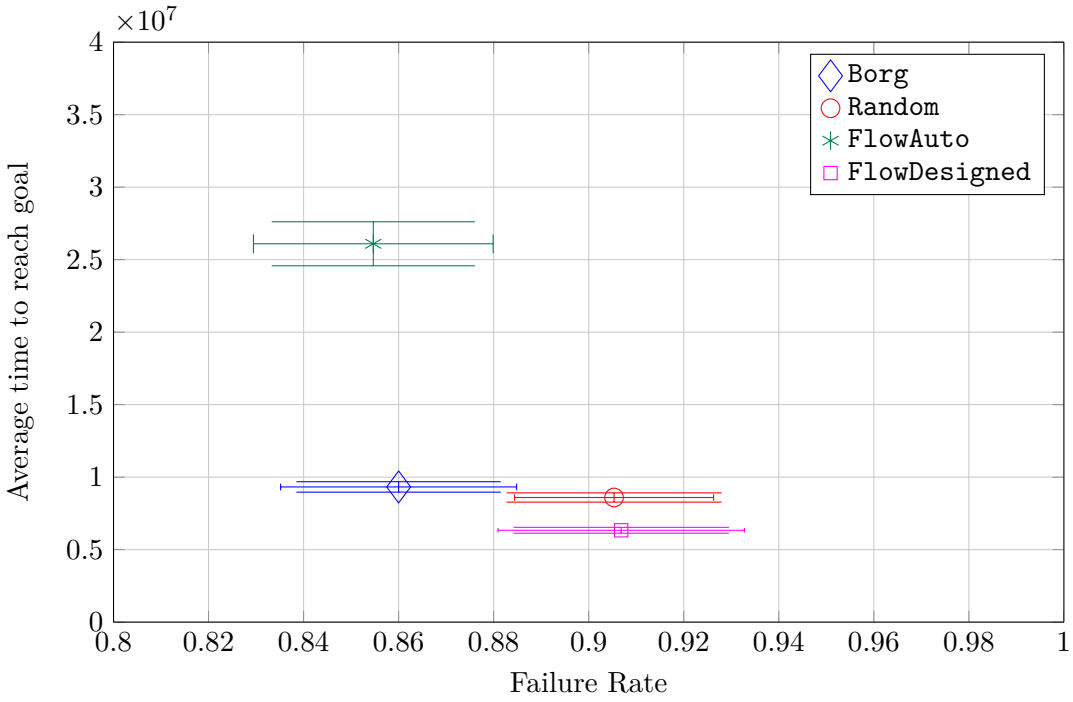

Figure 5.3: Average time to reach goal and failure rate (with 95% confidence intervals) for agents following four different metacontrol policies.

.

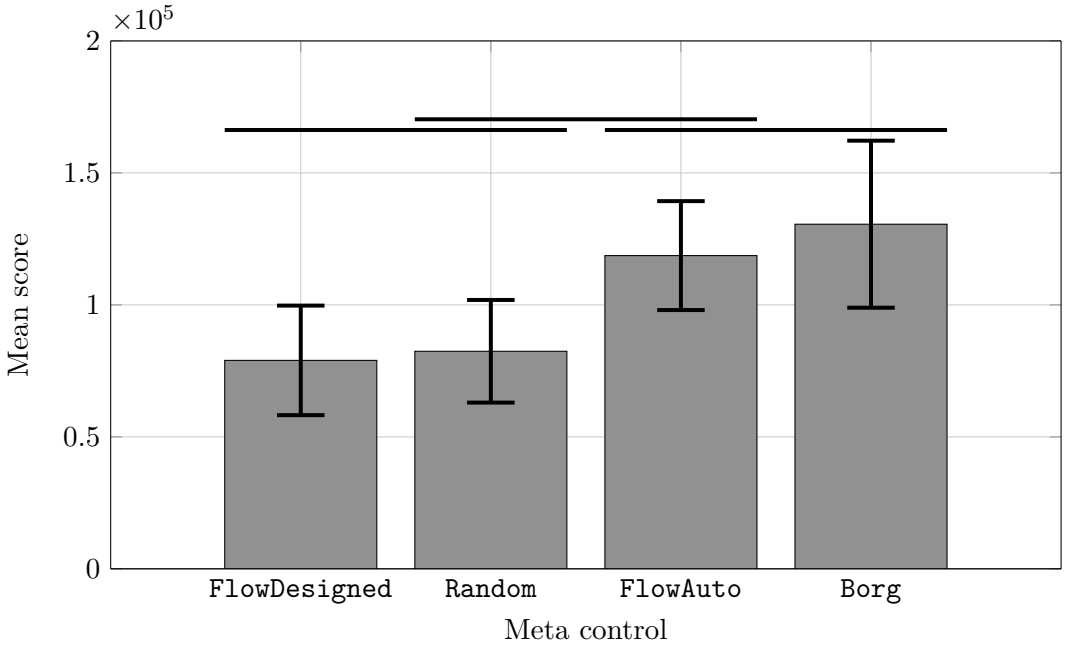

Figure 5.4: Mean score (±95% confidence intervals) for agents following four different metacontrol policies. Lines over bars indicate groups which were not significantly different (permutation F-test with Holm-Bonferroni corrected *post-hoc*).

5.3.2 Follow-up Experiments

Surviving `FlowAuto` agents took almost 20 million turns longer to reach the goal compared to agents using other metacontrols. This may have been because they spent an average of 8.4 ± 2.5 million turns on level 7 compared to a grand mean of 0.5 ± 0.3 million turns per level (for `FlowAuto` agents). By contrast the average turns per level 7 for `borg` agents is 0.66 ± 0.08 million turns.

This unusually long stay at level 7 coincided with a decrease in mined complexity for character strength (STR) from 18 at level 7 to 17 at level 8 (Figure 5.5). We tried removing this dip in complexity by modifying the complexity of levels before level 8. In a 300 trial experiment the resulting metacontrol, `FlowAuto`*, was no different from the `FlowDesigned` metacontrol on any measure (Table 5.4; 300 trials).

On the other hand, "pinning" the `random` metacontrol to level 7 for 5 million turns (i.e., preventing the metacontrol from descending to level 8 until at least 5 million turns after first entering level 7) decreases failure rate by 8% although the difference in score is not significant (Table 5.5; 300 trials).

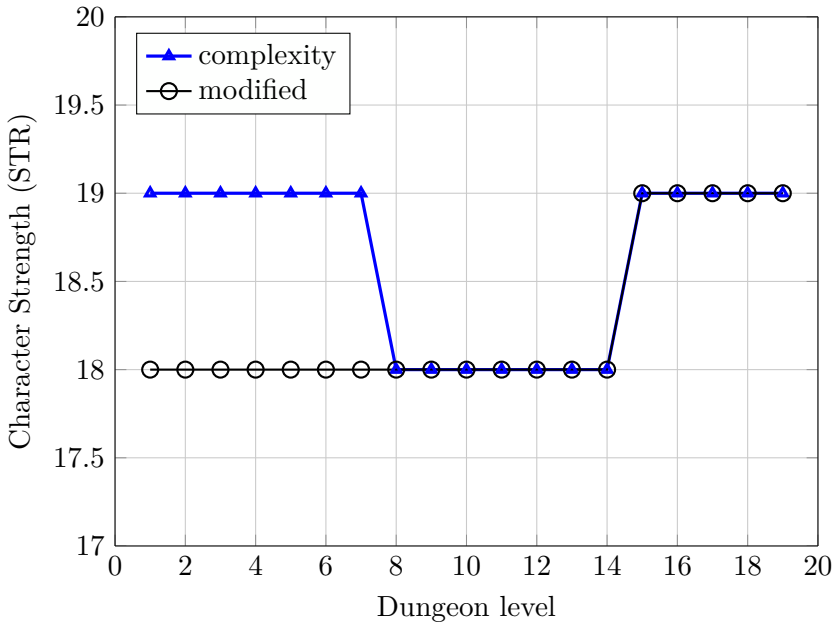

Figure 5.5: Strength (STR) required to reach the goal as a function of level as found by our complexity mining (triangles). The circles represent our modified complexity in the follow-up experiment.

Table 5.4: Mean turns to reach the goal, failure rate, and score of the `FlowAuto`* and `FlowDesigned` metacontrols with 95% confidence intervals.

	Mean turns [millions]	Failure rate [%]	Score [thousands]
`FlowAuto`*	6.14 ± 0.16	92.2 ± 3.0	61 ± 28
`FlowDesigned`	6.33 ± 0.20	90.7 ± 2.6	79 ± 21
absolute difference	0.18	1.5	18

Table 5.5: Mean turns to reach the goal, failure rate, and score of the `random` and pinned `random` metacontrols with 95% confidence intervals.

	Mean time [millions]	Failure rate [%]	Score [thousands]
pinned `random`	13.02 ± 0.35	82.0 ± 4.3	140 ± 40
`random`	7.97 ± 0.47	90.0 ± 3.4	88 ± 30
absolute difference	5.05	8	52

5.3.3 Discussion

The results indicate that a flow-seeking metacontrol can have an effect on performance and show improvement over a random metacontrol. The `FlowDesigned` metacontrol was faster but no less reliable than `random` and the `FlowAuto` metacontrol was more reliable and higher scoring, although slower. However, the existing `borg` metacontrol seems the best overall, it is more reliable and higher scoring than `FlowDesigned` and faster but no less reliable than `FlowAuto`.

The `FlowAuto` agents showed the most improvement over the `random` metacontrol but there is reason to suspect that this improvement is due merely to a decrease in the mined complexity of a single skill (STR) from level 7 to level 8. We speculate that this decrease does not represent the actual complexity profile in *Angband* since initial values of all skills were fixed. It is therefore possible that our mining process fails to accurately find the minimum skill required if the initial values set are higher than the actual complexity of those levels. Once effects that decrease the skills (e.g., attacks that drain statistics or destroy equipment) come into play the mined complexity will dip if the actual complexity is lower than what the agent initially had.

In fact, as the flow-seeking agents proved to be sensitive to the mined complexity there is reason to suspect that they would be similarly impacted by any other inaccuracies in mined complexity. We speculate that a better mining process, one that more reliably determines the minimum skills required by the ground policy on each level, could allow the flow-seeking metacontrol to compete with the borg in *Angband*. This may require a better mining algorithm or even a different definition of complexity (Section 6.2).

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this work, we presented an algorithm for metacontrol. We assumed a sequential decision-making in an environment that can be divided into levels of increasing complexity, and the goal is to reach the final level as quickly as possible. We based our algorithm on the psychological state of *flow*; specifically on the concept of matching the skills of the agent to the complexity of a level.

First, we showed how, for a given set of skills, data from probe agents can be used determine complexity by finding the minimum skill required of agents on a given level to reach the goal. We then presented a simple mathematical model of the degree of flow that maximizes when the skills of the agent match the mined complexity. This model led to a metacontrol policy that seeks to maximize degree of flow to find the state closest to flow.

We gave an example of how the flow-seeking metacontrol can be applied to a simple, single-skill environment and then tested it in the role-playing game *Angband*. We used both a hand-selected set of skills and an automatically selected one. Flow-seeking agents with the hand-selected skill set had the fastest average time to reach the goal but were no more reliable or higher scoring than a random metacontrol and worse than the existing hand-coded metacontrol. Flow-seeking agents using an automatically selected skill set proved to be the slowest; but reliability and score were the same as the existing hand-coded agents.

We also found the flow-seeking metacontrol to be sensitive to variations in the mined complexity; especially to a decrease in a skill value from one level to the next. This led us to speculate that a different scheme for complexity mining could perform better.

6.2 Directions for Future Work

The results of our algorithm in *Angband* indicated that complexity mining plays an important role. The technique presented in this thesis is based on the minimum skills required by probe agents to reach the goal. Generally it will accurately estimate the actual complexity, as we define it, only if exploration of probe agents is sufficient. Other methods of estimating complexity, such as genetic search [14] where agents evolve a complexity profile, could be evaluated in future work.

Another line of future work will explore removing some the assumptions used in our current estimation of complexity. For example, we assumed for each feature a higher value means better survival, regardless of other feature values. However, skills may be correlated such as a character's armour weighting them down; thus reducing their speed. A player can invest in either heavy armour or high speed to prevent damage from hits and increase survivability. Agents relying on a per-component minimum, however, would target neither speed nor armour since the minimum for armour would come from the fast characters and the minimum for speed would come from the heavily armoured characters. A potential solution is to use automated clustering techniques to first define the modes of the skill profiles needed to operate at a level. Then one may take the minimum of abilities *within* each clusters to arrive at the complexity for each mode.

Another aspect our algorithm neglects is *skill weighting*. Our degree of flow is based on matching all components of the skill and complexity vectors equally. This does not take into account the potential for some features to be more important than others for defining the complexity of a level. An example of this are hit points in *Angband*. With low hit points the character can die to a low level monster. Even if all other skills match the desired values for a level perfectly when hit points are low surviving the level becomes difficult. This problem could be corrected by weighting skills such that something important (e.g., hit points) would receive a higher weight.

Another potential direction for research is the applicability of our flow model. Our model is based on the concept of matching agent's skills to complexity but it is not known whether or when doing this maximizes performance in an environment. There is no theoretical proof that the assumptions we made on the environment are necessary or sufficient to ensure the flow-seeking agents reach optimal performance. Flow would likely not be applicable to all environments. For example, in path-

finding a flow-seeking agent may feel that climbing a mountain is just the right complexity for his skills although the easier path through a mountain-pass is a more optimal solution.

Finally, the mathematical model of flow used in this thesis has not been psychologically validated. Other models of flow exist [30, 45, 23] and although studies [31] confirm that flow is linked to the match between skills and difficulty, to the best of our knowledge no experiments have been conducted to validate our model. User studies, using established measures of flow, could reveal if the flow-seeking metacontrol improves experience of flow for human players.

Bibliography

[1] A Players' Guide to Angband., 2015. URL http://rephial.org/help/guide. [Web. 22 Apr. 2015].

[2] Michael L Anderson and Tim Oates. A review of recent research in metareasoning and metalearning. *AI Magazine*, 28(1):12, 2007.

[3] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[4] D.P. Bertsekas. *Dynamic Programming and Optimal Control*. Number v. 1 in Athena scientific optimization and computation series. Athena Scientific, 1995. ISBN 9781886529120. URL https://books.google.ca/books?id=RjGQQgAACAAJ.

[5] Blai Bonet and Hector Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 1233–1238, 2003.

[6] Vadim Bulitko. Flow for Meta Control. *CoRR*, abs/1407.4, 2014. URL http://arxiv.org/abs/1407.4709.

[7] Vadim Bulitko and Matthew Brown. Flow Maximization as a Guide to Optimizing Performance: A Computational Model. *Adv. in Cognitive Systems*, 2(239-256), 2012.

[8] Massimo Carli, Antonella Delle Fave, and Fausto Massimini. The quality of experience in the flow channels: Comparison of italian and us students. 1988.

[9] Michael Cox and Anita Raja. Metareasoning: A manifesto. *BBN Technical*, 2007.

[10] M Csikszentmihalyi. *Beyond Boredom and Anxiety*. The Jossey-Bass behavioral science series. Jossey-Bass Publishers, 1975. ISBN 9780875892610.

[11] Alex Cutler, Andy Astrand, Sean Marsh, Geoff Hill, Charles Teague, and Charles Swiger. Angband. http://rephial.org/, 1990.

[12] Peter Dayan and Geoffrey E Hinton. Feudal Reinforcement Learning. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 271–278. Morgan Kaufmann, 1993.

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

[14] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. ISBN 0201157675.

[15] M A Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

[16] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The {WEKA} Data Mining Software: An Update. *{SIGKDD} Explorations*, 11(1):10–18, 2009. URL `http://www.sigkdd.org/explorations/issues/11-1-2009-07/p2V11n1.pdf`.

[17] Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical Solution of Markov Decision Processes using Macro-actions. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 220–229, 1998. ISBN 1-55860-555-X.

[18] J.J. Higgins. *An Introduction to Modern Nonparametric Statistics*. Duxbury advanced series. Brooks/Cole, 2004. ISBN 9780534387754. URL `https://books.google.ca/books?id=vhmFQgAACAAJ`.

[19] Jesse Hoey, Pascal Poupart, Axel von Bertoldi, Tammy Craig, Craig Boutilier, and Alex Mihailidis. Automated handwashing assistance for persons with dementia using video and a partially observable Markov decision process. *Computer Vision and Image Understanding*, 114:503–519, 2010. ISSN 10773142. doi: 10.1016/j.cviu.2009.06.008.

[20] R A Howard. *Dynamic Probabilistic Systems: Semi-Markov and decision processes*. Series in Decision and Control. Wiley, 1971. ISBN 9780471416661. URL `http://books.google.ca/books?id=vuZQAAAAMAAJ`.

[21] Robin Hunicke and Vernell Chapman. AI for dynamic difficulty adjustment in games. *Challenges in Game Artificial Intelligence AAAI ...*, pages 91–96, 2004. doi: 10.1145/1178477.1178573. URL `http://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-019.pdf`.

[22] Leslie Pack Kaelbling. Hierarchical Learning in Stochastic Domains: Preliminary Results. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 167–173. Morgan Kaufmann, 1993.

[23] Martin Klasen, René Weber, Tilo T J Kircher, Krystyna A Mathiak, and Klaus Mathiak. Neural contributions to flow experience during video game playing. *Social Cognitive and Affective Neuroscience*, 7(4):485–495, 2012. doi: 10.1093/scan/nsr021.

[24] Andrey Kolobov, Mausam, and Daniel S Weld. A Theory of Goal-Oriented MDPs with Dead Ends. *CoRR*, abs/1210.4, 2012.

[25] Giel Van Lankveld and Pieter Spronck. Difficulty Scaling through Incongruity. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 228–229, 2008. ISBN 9781577353911.

[26] W. S. Lovejoy. Computationally Feasible Bounds for Partially Observed Markov Decision Processes, 1991. ISSN 0030-364X.

[27] Brian Magerko, Brian S. Stensrud, and Lisa Scott Holt. Bringing the Schoolhouse Inside the Box - A Tool for Engaging Individualized Training. In *25th Army Science Conference*, 2006.

[28] Leonard A Marascuilo. Large-sample multiple comparisons. *Psychological bulletin*, 65(5):280, 1966.

[29] Patrick L Mayers. *Flow in adolescence and its relation to school experience.* PhD thesis, ProQuest Information & Learning, 1978.

[30] Giovanni B Moneta. On the measurement and conceptualization of flow. In *Advances in flow research*, pages 23–50. Springer, 2012.

[31] Giovanni B. Moneta and Mihaly Csikszentmihalyi. The Effect of Perceived Challenges and Skills on the Quality of Subjective Experience. *Journal of Personality*, 64(2):275–310, June 1996. ISSN 0022-3506. doi: 10.1111/j.1467-6494.1996.tb00512.x. URL `http://doi.wiley.com/10.1111/j.1467-6494.1996.tb00512.x`.

[32] Jeanne Nakamura. Optimal experience and the uses of talent. 1988.

[33] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*, pages 1043–1049. MIT Press, 1998.

[34] Ronald Edward Parr. Hierarchical Control and Learning for Markov Decision Processes, 1998.

[35] Chris Pedersen, Julian Togelius, and Georgios N. Yannakakis. Modeling player experience in Super Mario Bros. In *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, pages 132–139, 2009. ISBN 9781424448159. doi: 10.1109/CIG.2009.5286482.

[36] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 1025–1030, 2003. ISBN 1045-0823.

[37] Nicholas Roy and Geoffrey J Gordon. Exponential Family PCA for Belief Compression in POMDPs. In *Advances in Neural Information Processing Systems*, pages 1635–1642, 2002. ISBN 0262025507. URL `http://machinelearning.wustl.edu/mlpapers/paper_files/CN16.pdf`.

[38] J Schmidhuber. Curious model-building control systems. In *Neural Networks, 1991. 1991 IEEE International Joint Conference on*, pages 1458–1463 vol.2, November 1991. doi: 10.1109/IJCNN.1991.170605.

[39] Jürgen Schmidhuber. A Possibility for Implementing Curiosity and Boredom in Model-building Neural Controllers. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, pages 222–227, Cambridge, MA, USA, 1990. MIT Press. ISBN 0-262-63138-5.

[40] Satinder Pal Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8:323–339, 1992. ISSN 08856125. doi: 10.1007/BF00992700.

[41] Jan Storck, Sepp Hochreiter, and Jürgen Schmidhuber. Reinforcement Driven Information Acquisition In Non-Deterministic Environments, 1995.

[42] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, 1998.

[43] Richard S. Sutton, Doina Precup, and Satinder Singh. Intra-option learning about temporally abstract actions. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)*, pages 556–564, 1998. ISBN 1558605568.

[44] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(99)00052-1.

[45] René Weber, Ron Tamborini, Amber Westcott-Baker, and Benjamin Kantor. Theorizing flow and media enjoyment as cognitive synchronization of attentional and reward networks. *Communication Theory*, 19:397–422, 2009. ISSN 10503293. doi: 10.1111/j.1468-2885.2009.01352.x.

[46] Adam White. Borg Winners, 2015. URL http://www.innovapain.com/borg/borg-winners/. [Online; accessed 10-March-2015].

[47] Andrew P White. Angband Borg. www.innovapain.com/borg/, 1995. URL www.innovapain.com/borg/.

[48] Steven D Whitehead and Dana H Ballard. Active Perception and Reinforcement Learning. *Neural Computation*, 2(4):409–419, December 1990. ISSN 0899-7667. doi: 10.1162/neco.1990.2.4.409. URL http://dx.doi.org/10.1162/neco.1990.2.4.409.

[49] Georgios N. Yannakakis, Henrik Hautop Lund, and John Hallam. Modeling children's entertainment in the playware playground. In *Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games, CIG'06*, pages 134–141, 2007. ISBN 1424404649. doi: 10.1109/CIG.2006.311692.

[50] A Zook and M O Riedl. Temporal Game Challenge Tailoring. *Computational Intelligence and AI in Games, IEEE Transactions on*, PP(99):1, 2014. ISSN 1943-068X. doi: 10.1109/TCIAIG.2014.2342934.