Proceedings of the Fifteenth AAAI Conference on Artificial
Intelligence and Interactive Digital Entertainment (AIIDE-19)

Automatic Abstraction and Refinement for Simulations with Adaptive Level of Detail

Michelangelo Diamanti,[1] **David Thue**[1,2]
[1]Department of Computer Science, Reykjavik University, Menntavegur 1, 101, Reykjavik, Iceland;
[2]School of Information Technology, Carleton University, 1125 Colonel By Drive, Ottawa, ON, K1S 5B6, Canada
michelangelo17@ru.is, david.thue@carleton.ca

Abstract

Optimizing the level of detail of an interactive simulation involves maximizing its perceived scope while minimizing the computational resources that are required to maintain it. Using varying levels of detail is common in computer graphics, but the challenges of doing so in simulations remain substantially less explored. The interactive simulations of video games often govern the behaviour of intelligent agents in the environment, and such behaviours can take substantial computational resources to maintain. As the ambitions of designers and players demand larger and more complex simulations, new strategies are needed to disassociate the perceived scope of a simulation with its computational needs. To this end, we propose a way to automatically adjust between different levels of detail in an interactive, narrative planning context, while simultaneously identifying and visualizing the elements that can currently be perceived.

1 Introduction

For many years, the graphical components of video games have often been targeted as a crucial point of optimization. Every modern graphics engine employs a wide set of techniques aimed at automatically adjusting the level of detail of the rendered objects, to maximize the game's performance while providing rich visual details. Video games, however, involve more than only graphics; behind almost everything that happens in a game's environment, there is an algorithm that decides its behaviour. Although optimizing the simulations that underpin video games is highly important, the concept of adaptive level of detail has received relatively little attention in this context, and particularly in academia. As a result, recent advances on this front have been limited, as industry-focused solutions have remained tightly bound to the games for which they were made (e.g., *Total War – Rome II* (The Creative Assembly 2013; Beacco, Pelechano, and Andújar 2016) or *Assassin's Creed Unity* (Ubisoft Montreal 2015; Cournoyer and Fortier 2015)).

There are at least two points that make adjusting simulation level of detail (LoD) more difficult than optimizing graphics. First, while the way that graphics are represented has become relatively fixed, there are many different ways to model the behavioural simulation of a game's environment. When a simulation is modelled in a new way, it requires a new way to adjust its level of detail. This greatly hinders the re-usability of LoD self-adjustment techniques across different implementations. Second, since simulation LoD deals with semantics rather than visual props, there is no easy way to find which parts are best to optimize, nor which metrics can serve as good indicators for switching the level of detail.

In this work, we seek an approach that can be employed in many different scenarios to adjust the level of detail of various components of a simulation. To this end, we introduce a system that helps the user design simulations that are capable of self-adjusting their level of detail. The system consists of three components:

- A **framework for simulations** that allows the differentiation of multiple LoDs, accounts for multiple parallel agents, and can be easily ported to other systems.
- An **algorithm for detecting LoD transition points** that estimates the observability of each region that is being simulated in the environment, with regard to the player's perception of the world.
- An **algorithm for adjusting the level of detail** according to changes in the observability metric. When the metric is above a certain threshold, the algorithm parses each previous abstract simulation step into a series of more granular steps, whose combined effects lead to an equivalent state. Analogously, when a region's observability decreases below a certain level, the algorithm drops all the aspects of the simulation that are absent from next lower level of detail, while still maintaining the aspects that remain.

1.1 Background and Motivation

Adjusting the simulation level of detail of a video game is a challenging problem. Many modern video games strive to fill their environments with detailed agents and features, toward enabling ever deeper levels of player immersion. At the same time, the complex simulations of games often have many connected parts that influence one another with varying degrees of magnitude, and fully computing these simulations can require a prohibitive amount of resources. Finally, this tension between maximizing complexity and minimizing resource usage is further complicated by the presence of a player, and particularly in games that focus on their narrative. In such games, both the state of the simulated world and the state of the story must be maintained with consistency, so as not to challenge the player's suspension of disbelief. Simulating narrative-focused world spaces thus involves striking a balance between minimizing the use of resources and maintaining a consistent world.

Most video game simulations belong to one of two categories. In one category, two levels of simulation detail are used: game elements located near the player are fully simulated, while elements that are out of range are not simulated at all. For example, in open-world games like *Skyrim* (Bethesda Game Studios 2011), the player can easily gain the impression that nothing in the world advances without them being present. This approach works well to lower the game's overall resource requirements, but when different elements should reasonably be affected by one another, the lack of simulation of one might create inconsistencies in the simulation of the other. For this reason, other video game simulations consider only a single level of detail, spending the same amount of resources even for elements that are not perceived by the player (e.g., *Starcraft II* (Blizzard Entertainment 2010)). We aim to strike a balance between these two alternatives that retains the benefits of both: avoid the computation of certain details to minimize resource use, but ensure the computation of sufficient details to maintain a consistent simulation.

At a high level, we approach this challenge by distinguishing between three kinds of details: (i) those whose computation must be *immediate*, (ii) those whose computation can be *delayed*, and (iii) those whose computation can be *avoided*. The main advantages of this approach are twofold:

- **Avoiding Unnecessary Details**: If the simulation never reaches a state that compels the computation of details that were delayed, then those details become avoided and the simulation is spared their computation.
- **Computation Scaffolding**: If the simulation *does* reach a state that compels the computation of delayed details, then this computation can be constrained by the structure of what had occurred at a more abstract level of detail. This greatly reduces the size of the problem.

The following definitions will aid in presenting our work. A *domain* is a model that can be used to simulate a given phenomenon. It contains all the constructs that can be used to describe an instance of the environment, as well as *transition model* that governs how the simulation progresses between states. Given two domains, δ_1 and δ_2, we say that δ_1 is a *consistent abstraction* of δ_2 if every possible state that is reachable by applying transitions defined in δ_1 is also reachable by applying transitions defined in δ_2. Given a sequence of n domains $\Delta = \langle \delta_1, \ldots, \delta_n \rangle$, if each $\delta_i \in \Delta$ is a consistent abstraction of $\delta_{i+1} \in \Delta, \forall i \in [1, n-1]$, then we say that Δ is a *multi-level domain*, and use δ_i to denote the domain at level of detail i. The most detailed domain in Δ is δ_n, at level of detail n.

2 Problem Formulation

The problem that we aim to solve is as follows. Given a multi-level domain for an interactive, narrative simulation, how can we:

1. Translate a given detailed state into a more abstract state;
2. Translate a given abstract state into a more detailed state; and
3. Use knowledge of what the player can observe to decide when the level of detail should be changed?

The simulated story should be **consistent** at all times so as not to challenge the user's suspension of disbelief. The simulation should provide an **adequate level of detail** to account for every aspect of the world that the user can perceive. The switch in level of detail should take place according to an objective estimate of the **user's world perception**. Adaptive level of detail should result in **saving resources** compared to always simulating at the highest possible level of detail.

3 Related Work

There have been several attempts to adjust the level of detail of the behavioural component of simulations, and many of them have focused on virtual humans. O'Sullivan et al. (2003) introduced a way to adjust the LoD of simulations with regards to three aspects: geometry (graphics), motion, and behaviour such as spoken and unspoken cues. Paris, Gerdelan, and O'Sullivan (2009) discussed the simulation of a large crowd of virtual humans, including finding paths and avoiding obstacles. Niederberger and Gross (2005) presented a scheduling algorithm that divides the time allocated for computing the behaviour of each agent depending on its observability. They considered two types of behaviour: proactive (accurate, but expensive), and reactive (cheap, but only avoids inconsistencies).

Other approaches have sought to leverage the advantages of particular representations of the simulated world. For example, Brom, Šerý, and Poch (2007) presented a way to adjust the level of detail of both spatial and behavioural components of simulations. Their approach relied on the concept that, if spaces are organized hierarchically, they can be divided in multiple layers of abstraction. Osborne, Dickinson, and others (2010) focused on simplifying the LoD of simulations that are composed of hierarchically structured groups of intelligent agents. The authors simulated many separate groups of similar entities, comparable to a flock, where each group may contain an arbitrary number of agents.

Some literature has explored the question of when to switch the LoD. For example, Chenney, Arikan, and Forsyth (2001) presented an approach for decreasing the motion level of detail for objects that are not perceivable by the user. The authors define the term "proxy simulation" as the procedure responsible for managing the objects that are out of scope, with a lower level of detail. Kistler, Wißner, and André (2010) devised a formula for switching LoD based on distance from the camera and occlusion. Sunshine-Hill (2013) proposed to change the LoD using a more comprehensive formula, taking into account metrics of observability, memory, return time, and duration. All those values were used to estimate the inverse probability of incurring a break in realism, toward choosing the LoD.

Some authors have applied notions from AI planning in the context of simulations, similarly to what we do in this work. For example, Sacerdoti (1974) studied the difficulties of devising efficient plans for complex environments. The core concept of their work was that one can greatly increase the performance of classic planners by reasoning about the problem at hand on different levels of abstraction. The planner starts by differentiating between important parts of the problem and the ones that can be regarded as (trivial) details. In this regard, each level of abstraction can be thought of as a distinct planning domain. Plans are devised at high level and then refined to account for details at ground level.

Riedl and Young (2005), Li et al. (2014), Thue et al. (2016), and Robertson and Young (2019) all described types of narrative planning that leveraged a player's limited knowledge about the story's world. While they exploited this limitation to better meet a set of narrative constraints, we use it to reduce the complexity of simulating the story's world.

Several of these approaches are confined within their own scopes (limiting their use for other scenarios), and those that are more general often put a limit on the number of levels of detail that can be used. Differently from these methods, our work seeks a way to adapt the granularity of a simulation that supports an arbitrary number of detail levels, and in a way that offers general support for narrative reasoning in the context of a simulated world (Ware and Young 2014).

4 Proposed Approach

We first introduce and describe the language that we used to model the simulation and the resulting framework that we implemented. We then explain how we used the framework to model a simulation that can adjust its level of detail, and present an algorithm for increasing or decreasing the level of detail. Finally, we explain how we estimated observability and how we employed that measurement to switch between the various levels of detail.

4.1 The Planning Domain Definition Language

The Planning Domain Definition Language (PDDL) is an attempt to standardize Artificial Intelligence (AI) planning languages (Ghallab et al. 1998). A classic problem expressed in PDDL has two main components:

- A **domain description** contains the elements that are common across all variations of a problem that we are modelling, including *predicates* that can be used to describe facts about the world and *actions* that each agent can perform. See Figure 1 (left). The actions defined in the domain define its transition model (how the world moves between states) since their preconditions and effects determine which actions can be performed in which states.
- A **problem description** describes a specific problem to be solved within the given domain. It contains all the *entities* that are part of the problem and the *relations* (predicates instantiated with entities) whose conjunction describes the world's initial *state*. See Figure 1 (right).

Domain Description	Problem Description
Types:	*Entities:*
ROVER, WAYPOINT	ROVER rov1
Predicates:	ROVER rov2
AT(ROVER, WAYPOINT)	WAYPOINT way1
CONNECTED(WP, WP)	WAYPOINT way2
Actions:	WAYPOINT way3
MOVE(rover, WP1, WP2)	
Preconditions:	*Relations:*
rover AT WP1	rov1 AT way1
WP1 CONNECTED WP2	rov2 AT way2
Postconditions:	way1 CONNECTED way2
rover AT WP2	way2 CONNECTED way3

Figure 1: A possible representation of a PDDL problem. Usually PDDL would include a goal state definition in the domain, but we use PDDL for simulation – not planning.

The actions defined in the domain of a PDDL problem do not distinguish which (if any) of their parameters is the one that actually performs the action, since simple planning problems only have one entity that can perform actions. Given our desire to model environments with more than one active entity, we modified the PDDL standard to additionally specify the *subject* and *object* (if any) of each action.

4.2 Simulation Using the PDDL Framework

We represented each level of the simulation using a directed tree structure. A directed tree is a graph in which any two vertices are connected by exactly one path (ignoring edge directions). Each node represents a world state, while each edge represents an action from the domain. Figure 2 depicts a possible unfolding of the game tree after 3 steps. Although this representation has many branches, once an action is selected, the ones that start with other actions are discarded.

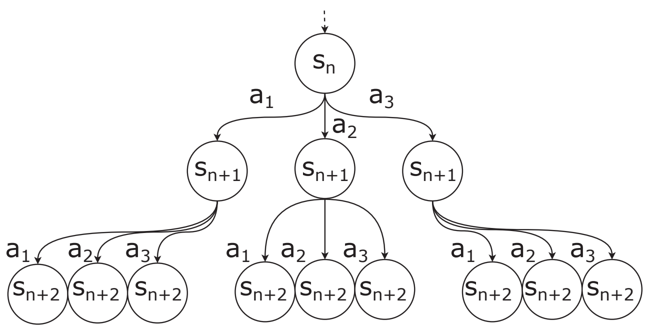


Figure 2: PDDL simulation represented in a tree structure. The nodes of the tree are the states, while the edges are the actions performed to transition from one state to the next.

As shown in Figure 3, the simulation process iterates between planning and visualization: the simulator chooses an action from the domain's transition model according to the current world state and then instructs the visualizer to perform it. The visualizer is responsible for showing the action to the player as well as detecting any errors or player actions that may occur during the process. After the visualization cycle is completed, the simulator will receive an acknowledgement that will inform it about the outcome of the action: if it was successfully visualized, the simulation moves forward to consider a new action; otherwise, it rolls back.

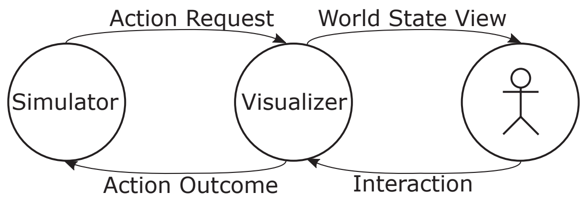


Figure 3: Representation of the simulation loop.

To choose an action from the current domain's transition model, the simulator must first compute a list of actions that are allowed in the current world state: an action is deemed performable if the entities in its parameter are part of the entities of the world state and all its preconditions are satisfied by relations in the current world state. Since part of our approach involves delaying the computation of certain world state details and then revisiting that computation later, it is important to store at least a partial *history* of the states that have been traversed by the simulation. Since we represent the simulation using a tree, it is enough to save the last visited node for each level of detail (see Figure 4).

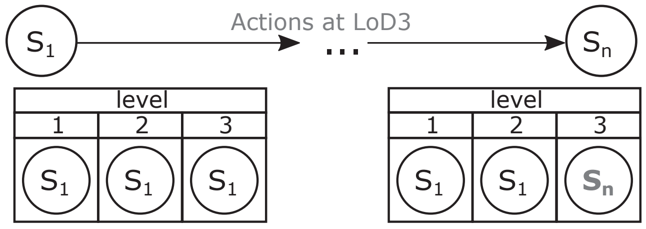


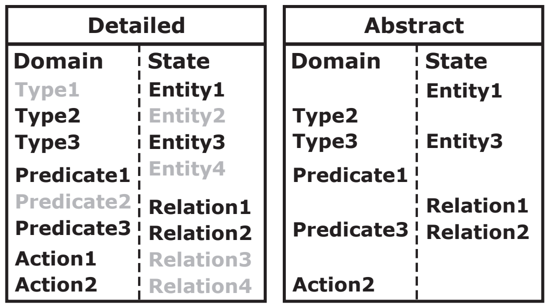
Figure 4: A visualization of how a separate history is maintained at each LoD. If the simulation starts in s_1 at LoD3 and actions occur leading to state s_n, then s_n will be stored as the last observed state at LoD3.

4.3 Level of Detail Adjustment

To delay or avoid the computation of different details in the world state, we devised an algorithm to translate information between two levels of a multi-level domain. We refer to the process of decreasing detail as *abstraction*, and the process of increasing detail as *refinement*. We describe each in turn.

Abstraction. Abstraction is the process of translating information from a detailed model to a more abstract one. Performing abstraction means that we drop some of the details of the current world state so that we can reduce the size of the state space, thus reducing required resources in terms of memory and computation time.

Figure 5 shows an example of how abstraction is performed. First, the domain at each level of detail (columns 1 and 3) and the detailed state (column 2) are given as inputs;



Detailed	
Domain	**State**
Type1	Entity1
Type2	Entity2
Type3	Entity3
Predicate1	Entity4
Predicate2	Relation1
Predicate3	Relation2
Action1	Relation3
Action2	Relation4

Abstract	
Domain	**State**
	Entity1
Type2	
Type3	Entity3
Predicate1	
	Relation1
Predicate3	Relation2
Action2	

Figure 5: An example of the outcome of abstraction. Gray highlights in column 1 indicate elements of the detailed level's domain that are removed by the algorithm in the more abstract level's domain (column 3). Similar highlights in column 2 indicate which entities or relations are removed from the detailed state while creating the abstract state.

the goal is to produce an abstract state (column 4). Next, the types and predicates that are present in in the detailed domain but *missing* from the abstract domain are identified (shaded gray in the figure). Since each entity depends on a type and each relation depends on a predicate, the missing elements can act as a filter when determining the target abstract state. Starting with a copy of the detailed state, an abstract state is produced by removing every entity whose type is missing and every relation whose predicate is missing. The result is shown in column 4 of the figure. It is important to specify that each time we perform abstraction, we replace the old abstract branch with a new one containing only the current abstract node as the root. The old branch is useless because we have a more detailed version saved in the structure of Figure 4. The system then continues from the root node to advance the simulation at the abstract level.

Refinement. Refinement is the process of translating information from an abstract model to a more detailed model. This function is performed when the player triggers the switch from an abstract simulation to a more detailed one. We first roll back the simulation until we hit the root node. Since every time we perform abstraction we discard the current branch and only store the last node in the table, the root node will correspond to the last observed state at the previous level. After that, we start traversing the simulation tree and pairing each abstract action with a list of detailed ones, whose cumulative effect produce an equivalent resulting node. To do that, we use a search algorithm whose inputs are the starting node at the detailed level and the desired node at the abstract level. The algorithm applies the detailed transition model to traverse the tree and tries to reach a node, at the detailed level, which is equivalent to the desired abstract one. An equivalent detailed node is one that contains all the effects of the abstract actions, while still also satisfying the constraints for the additional details.

Referring to Algorithm 1, we first roll back the simulation at the abstract level of detail until we hit the root node (lines 3-5). At this point, we have both the detailed and abstract version of a common node, from before the last abstraction

Algorithm 1: The subgoal search algorithm used to obtain a detailed node during refinement.

```
Input  : Leaf node at abstract level of detail
Output: Equivalent node at detailed level of detail
1  currentAbstract : Current abstract node;
2  lastDetailed : Last observed detailed node;
3  while not IsRoot(currentAbstract) do
4      currentAbstract ← Parent(currentAbstract);
5  end
6  nextAbstract ← Child(currentAbstract);
7  while nextAbstract ≠ null do
8      detailedPlan ← FindPlan(lastDetailed,
9                                nextAbstract);
10     lastDetailed ← Apply(lastDetailed,
11                              detailedPlan);
12     nextAbstract ← Child(nextAbstract);
13 end
14 return lastDetailed
```

step. Note that in line 6, we can traverse the tree as a list; after we commit the chosen action, the branching factor at every depth becomes one, and so each parent is left with only one child. We start traversing the simulation tree and pairing each abstract node with a list of detailed ones, until we reach a detailed node equivalent to the abstract one (lines 7-13). We do so by feeding the nodes at both levels of detail to a planner (line 8-9). This planner finds a sequence of actions - *a plan* - that leads the system from the old detailed state, to a new one, which is equivalent to the next abstract state. The plan is then applied to the last detailed state to obtain a new detailed state (lines 10-11). This process continues until we have a detailed plan for the whole abstract branch. The most basic version of our planner is a slight modification of the classic Breadth First Search.

Concurrency. Allowing multiple agents to perform actions at the same time could lead to conflicts, since the effects of two parallel actions could have inconsistent effects on the world state. For this reason, we restricted the execution of parallel actions to only allow those whose effects on the world state remain invariant with respect to the order in which the actions are performed. To do so, we compute the possible permutations of a given set of actions and compute their cumulative effect on the simulation by applying each action to the current world state in each permutation's order. If the application of every permutation results in the same state, the actions can be performed in parallel. Note that they are not run by different threads – they are applied to the state in the same time step.

Synchronization. Another issue to consider when dealing with joint actions is their effect during the refinement process. In particular, when joint actions are refined, their refined actions might need to be synchronized. For example, consider an action *give* which, to simplify computation at LoD2, does not require the involved entities to be at the same location. If the *give* action at LoD3 requires the agents to share the same location, then the refined plans would need to take this requirement into account by making the agents meet. If the two agents will take different amounts of time (as executed in the game) to reach the meeting point, the first one to arrive will need to wait. We solve this issue by finding synchronization points if needed, and scheduling the execution of actions in such a way that every time constraint is satisfied. In certain cases (such as the given example), this involves the introduction of *idle* actions.

4.4 Level of Detail Switcher

In addition to knowing *how* to switch between different levels of detail, it is important to understand *when* each change should occur. We devised a function that outputs an estimate of a simulation region's observability based on two indicators of the player's ability to perceive the environment:

- **Proximity** measures the inverse of the distance between the player and the region.
- **Visibility** estimates whether the player's vision of the region is occluded by some obstacle.

The function computes a weighted average between these indicators (where $n = 2$):

$$\frac{\sum \prod_{i \leftarrow 0}^{n} \phi_i w_i}{\sum_{i \leftarrow 0}^{n} w_i} \qquad \phi_i : indicator\ measure \quad w_i : indicator's\ weight$$

We chose to employ a weighted average function because it is both versatile and easy to tune. Each domain is related to a specific observability threshold, and the LoD of the simulation is adjusted accordingly.

5 Testbed and Demonstration

To demonstrate and evaluate our approach, we created a testbed that models the behaviour of two AI agents in a simulated environment. The agents do not have a precisely defined goal, which allows their behaviour to be simulated by applying the transition model of a given domain.

The scenario contains two rovers that can perform multiple actions: moving between waypoints, taking samples of the soil, dropping the sample on a specific dock, and snapping pictures of objectives. We modelled this environment in our PDDL framework as follows. First, we split the general problem into a multi-level domain with three levels, where each LoD's domain varied from the others (e.g., with different actions, predicates, or preconditions on actions).

First Level of Detail: The rovers are free to move among 9 existing waypoints as long as the source and target destination are connected. If a rover happens to be at a waypoint that contains a sample, it can collect it. Analogously, if an objective is visible from a waypoint, the rover can take a picture of it. One of the waypoints is marked as a dropping dock, where the rover can drop the collected sample. The two rovers both start at the same location.

Second Level of Detail: To the elements in the first LoD, we added a new predicate that indicates the presence of obstacles between waypoints. The resulting relation has a directional meaning: if there is an obstacle between waypoints x and y, the rover cannot go from x to y, but it can still go from y to x (e.g., rolling down a hill). There are 4 obstacles.

Third Level of Detail: To the second LoD, we added predicates to represent a battery on each rover, and we implemented a transition model that regulates the charge level of the batteries. We modified the available actions so that they cause the battery to transition correctly between charge levels. We added a new action that lets a rover charge its battery.

Suppose that the player starts in a setting of low observability, such that the observability estimate leads the system to run the simulation at LoD1. As three time steps pass, *rover one* visits waypoints 2, 6, and 8, while *rover two* visits waypoints 3, 5, and 4. At this point, if the player's observability increases past a given threshold, the LoD Switcher will transition to LoD2 and begin a refinement process:

- The system rolls back the simulation until it hits the root node. From there, it uses a planner to compute, for every set of parallel actions, an equivalent sequence that also accounts for the details at the new, more detailed level.
- The constraints of the new level of detail prevent *rover one* from going directly between waypoints 6 and 8 – there is an obstacle in between. The planner will thus return a detour, passing through waypoints 3 and 4 to reach the destination. The path for *rover two* remains clear.
- Since a single abstract action for *rover one* was refined into three detailed actions while *rover two*'s parallel action was refined into one, *rover two* will idle for two steps before performing its action.

After the refinement process is complete, sufficient details will have been computed to maintain simulation consistency. We employed the same process to go from level of detail 2 to 3 and account for the battery charge of the rovers.

Now, suppose that a large object appears in front of the player while the simulation is running at LoD3. This occludes the view of the player, causing the observability function to drop below the threshold of LoD2. The LoD Switcher triggers the abstraction process, and all of the relations related to the battery system are dropped from the current state. The system then continues at LoD2, saving resources without the player noticing the drop in LoD. Similarly, a transition from LoD2 to LoD1 would relieve the system of accounting for obstacles whenever the rovers move.

6 Evaluation and Discussion

To evaluate our approach, we computed the theoretical size of the state space for each of the LoDs by combining the various relations used to describe the instance of the problem with the actual entities that are part of the world. We also computed how many actions our planner considered at each step of its planning process (one value for each LoD), along with the amount of time that it took to do so. We performed all computations on a system with an Intel core i7-4700mq cpu @ 2.40GHZ, an NVIDIA GeForce GT 750M with GDDR5 2GB VRAM and 8GB DDR3 RAM. Times were averaged over a few subsequent runs.

LoD	Entities	Relations	State Space	Actions	Time (ms)
1	25	280	1.9E84	524	30
2	25	361	4.7E108	522	30
3	31	409	1.3E123	16770	1100

Table 1: Size of the state space at each level of detail. Expanded actions and related computational time.

Interestingly, Table 1 shows that we expanded more actions at LoD1 compared to LoD2, even if it should have more detail and thus more complexity. The reason is that the complexity is only represented by additional constraints: no new types or entities are added – only the preconditions of moving are changed to account for obstacles. Still, the number of actions increases greatly at the third level of detail, by a factor of 32. These results show that our system offers a clear performance advantage even when applied to simple environments such as the one in our demonstration, which fails to run at interactive frame rates at its highest LoD.

6.1 Limitations and Future Work

Our system has a few limitations. One is related to the limitations of modelling an environment with PDDL, as representing a large environment could require a large amount of effort. It is also difficult to make agents behave rationally, since we currently do not use PDDL's notion of goals. Another limitation is related to performance: while substantial resources can be saved at lower LoDs, the highest LoD used in our demonstration is still prohibitively expensive to run. Finally, the system could produce impossible plans: committing a refined set of actions could lead to a world state from which the abstract target could not be reached.

There are many ways in which we can improve the system. To expand beyond the limitations of PDDL and better represent agent actions, one might adopt a probabilistic model. One could also use more complex estimates of observability to switch between different LoDs, or consider other metrics such as those discussed by Sunshine-Hill (2013), Flores and Thue (2017), or Robertson and Young (2019). One might improve the overall performance of the refinement algorithm through caching systems or heuristics, toward expanding fewer states and reducing time and memory costs. To prove that the transition between LoDs is unnoticeable, we could conduct a user study.

Conclusion

We proposed a system that adjusts a simulation's level of detail in a narrative planning context. The system is based on a method that can translate between abstract and detailed world states using a multi-level planning domain, which is formalized using a language that is common in narrative planning (PDDL). Our approach leverages the idea that system resources can be saved by delaying or avoiding the computation of details that the user cannot perceive. The resulting system met our criteria for success: it saved resources while maintaining consistency thanks to its processes of abstraction and refinement, and it used an estimate of the player's perception to dynamically adjust the level of detail.

References

Beacco, A.; Pelechano, N.; and Andújar, C. 2016. A survey of real-time crowd rendering. *Computer Graphics Forum* 35(8):32 – 50.

Brom, C.; Šerý, O.; and Poch, T. 2007. Simulation level of detail for virtual humans. In Pelachaud, C.; Martin, J.-C.; André, E.; Chollet, G.; Karpouzis, K.; and Pelé, D., eds., *Intelligent Virtual Agents*, 1–14. Springer Berlin Heidelberg.

Chenney, S.; Arikan, O.; and Forsyth, D. A. 2001. Proxy simulations for efficient dynamics. In *Proceedings of Eurographics 2001*, 10 pages. Blackwell Publishers Ltd and the Eurographics Association.

Cournoyer, F., and Fortier, A. 2015. Massive crowd on Assassin's Creed Unity: AI recycling. Presentation at the Game Developer's Conference (GDC 2015). GDC Vault, UMB Tech.

Flores, L., and Thue, D. 2017. Level of detail event generation. In Nunes, N.; Oakley, I.; and Nisi, V., eds., *Interactive Storytelling*, 75–86. Springer International Publishing.

Ghallab, M.; Knoblock, C.; Wilkins, D.; Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Smith, D.; Sun, Y.; and Weld, D. 1998. Pddl - the planning domain definition language.

Kistler, F.; Wißner, M.; and André, E. 2010. Level of detail based behavior control for virtual characters. In Allbeck, J.; Badler, N.; Bickmore, T.; Pelachaud, C.; and Safonova, A., eds., *Intelligent Virtual Agents*, 118–124. Berlin, Heidelberg: Springer Berlin Heidelberg.

Li, B.; Thakkar, M.; Wang, Y.; and Riedl, M. O. 2014. Data-driven alibi story telling for social believability. In *Proceedings of the 2014 Foundations of Digital Games Workshop on Social Behavior in Games*, 8 pages.

Bethesda Game Studios. 2011. Skyrim. www.elder scrolls.com/skyrim.

Blizzard Entertainment. 2010. StarCraft II. starcraft2.com.

The Creative Assembly. 2013. Total War – Rome II. www.totalwar.com/games/rome-ii.

Ubisoft Montreal. 2015. Assassin's Creed: Unity. www.ubisoft.com/en-us/game/assassins-creed-unity.

Niederberger, C., and Gross, M. 2005. Level-of-detail for cognitive real-time characters. *The Visual Computer* 21(3):188–202.

Osborne, D.; Dickinson, P.; et al. 2010. Improving games ai performance using grouped hierarchical level of detail. In *Proceedings of the Third International Symposium on AI & Games, Daniela M. Romano and David C. Moffat (Eds.), at the AISB 2010*, 19 – 24. SSAISB.

O'Sullivan, C.; Cassell, J.; Vilhjalmsson, H.; Dingliana, J.; Dobbyn, S.; McNamee, B.; Peters, C.; and Giang, T. 2003. Levels of detail for crowds and groups. *Computer Graphics Forum* 21(4):733–741.

Paris, S.; Gerdelan, A.; and O'Sullivan, C. 2009. Ca-lod: Collision avoidance level of detail for scalable, controllable crowds. In Egges, A.; Geraerts, R.; and Overmars, M., eds., *Motion in Games*, 13–28. Springer Berlin Heidelberg.

Riedl, M. O., and Young, R. M. 2005. Open-world planning for story generation. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, 1719–1720. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Robertson, J., and Young, R. M. 2019. Perceptual experience management. *IEEE Trans. on Games* 11(1):15–24.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5(2):115 – 135.

Sunshine-Hill, B. 2013. Managing simulation level-of-detail with the LOD trader. In *Proceedings of Motion on Games*, MIG '13, 13:13–13:18. New York, NY, USA: ACM.

Thue, D.; Schiffel, S.; Árnason, R. A.; Stefnisson, I. S.; and Steinarsson, B. 2016. Delayed roles with authorable continuity in plan-based interactive storytelling. In Nack, F., and Gordon, A. S., eds., *Interactive Storytelling*, 258–269. Springer International Publishing.

Ware, S. G., and Young, R. M. 2014. Glaive: a state-space narrative planner supporting intentionality and conflict. In *Proceedings of the 10th Conference on Artificial Intelligence and Interactive Digital Entertainment*, 80–86. AAAI Press.