Characterizing and Optimizing Game Level Difficulty

Glen Berseth[1], M. Brandon Haworth[1], Mubbasir Kapadia[2] and Petros Faloutsos[1]
[1]York University
[2]Rutgers University

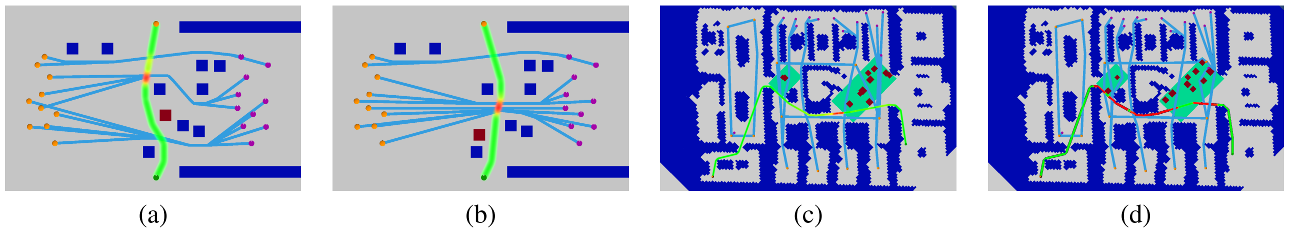


Figure 1: *Figures (a) and (b) illustrate how the location of an obstacle can affect a player's difficulty path (thicker shaded path). Difficulty is annotated with a gradient (hues, interpolated between light and dark, indicate low and high difficulty). Figures (c) and (d) result from optimizing for a desired difficulty on a parameterized scenario. The desired difficulty for (d) is much higher than for (c), which is reflected in the player's difficulty path.*

Abstract

Balancing the interactions between game level design and intended player experience is a difficult and time consuming process. Automating aspects of this process with respect to user-defined constraints has beneficial implications for game designers. A change in level layout may affect the available routes and subsequent player interactions for a number of agents within the level. Small changes in the placement of game elements may lead to significant changes in terms of the challenge experienced by the player on the path to their goal. Estimating the effect of this change requires that the designer take into account new paths of all interacting agents and how these may affect the player. As the number of these agents grow to crowd size, estimating the effect of these changes becomes grows difficult. We present a user-in-the-loop framework for tackling this task by optimizing enemy agent settings and the placement of game elements that affect the flow of agents within the level, with respect to estimated difficulty. Using static path analysis we estimate difficulty based on agent interactions with the player. To exemplify the usefulness of the framework, we show that small changes in level layout lead to significant changes in game difficulty, and optimizations with respect to the characterization of difficulty can be used to attain desired difficulty levels.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation I.6.8 [Simulation and Modeling]: Types of Simulation—Animation;

Keywords: procedural content generation, static analysis, optimization, game elements

1 Introduction

A significant challenge in game design is providing an engaging and entertaining user experience for players with skill levels that vary from beginner to expert. It is generally desired that the player be challenged but not overwhelmed by the game's difficulty. For certain types of games, adapting their difficulty to the user's skill level can be straightforward, such as the case of chess. For games where the player competes with non player characters (NPCs) in varying situations and environments, such as first person shooter games, this adaptation is generally not straightforward. Dynamically adapting the difficulty of such games to a player's skill level is an important research problem that is currently receiving significant attention. In this paper, we take a first step towards addressing this problem. Our goal is to provide an initial exploration of a framework that is simple, practical, and fairly general.

The paths a player chooses during gameplay are directly influenced by design choices made during the creation, or generation in procedural cases, of the game level. A small change such as the placement of a box or the initial configuration of an NPC can directly impact the user experience. These seemingly innocuous changes may lead to drastic changes in available cover, agent and player paths, behaviours, timing, etc. Simulating and analyzing all the possible actions and interactions that may occur in a game would be distinctly difficult. This is especially so in game scenarios where the player is afforded significant autonomy.

We address these challenges by presenting a user-in-the-loop framework for optimal NPC settings and placement of game elements that influence the flow of non-player characters in a game level and impact their resulting interactions with the player(s). Minor changes in level layout and the initial configurations of NPCs can dramatically impact the resulting difficulty of a level. Given an author-specified objective (e.g., desired level difficulty) and constraints on what aspects of the level can be changed, our framework can automatically generate plausible levels that satisfy the desired criteria. Our framework is general and can be easily integrated into existing game engines, and work for different game genres and AI systems.

The main contributions of this paper include: (a) an empirical measure of game level difficulty obtained by statically analyzing the game level for expected interactions between players, and NPCs (b) an optimization framework to automatically synthesize new game levels that satisfy the desired constraints on layout and difficulty. We demonstrate the potential of our framework by optimizing the placement of obstacles and certain parameters associated with the NPCs for levels that have been used in a few popular games.

2 Related Work

Game Level Evaluation and Generation. There are many methods for evaluating game level quality, such as [Liapis et al. 2013] which uses player challenge as a quality metric. The work in [Bauer and Popović 2012] uses Rapidly Expanding Random Trees (RRT) to sample a level's state space, then clusters the output tree of the RRT using Markov Clustering (MCL) to form a representative graph of the level. Recent simulation based analysis calculates expected scenario complexity with respect to a number of crowd simulation algorithms [Berseth et al. 2013].

Early work in [Güttler and Johansson 2003] outlines some spatial principals of level design and shows the effects of altering parts of the game level. The work in [Smith et al. 2012] uses constrained state-based search to generate new levels with some control over the style of the new level. Recent work in [Horswill and Foged 2012] uses constraint satisfaction propagation to generate game levels meeting specific requirements. Smith *et al.* [2010] moves in the direction of automating the whole game design process by forming a declarative representation of a game using answer-set programming.

Player Experience. The concept of *flow* [Csikszentmihalyi 1990] is a popular player experience measure and is used to characterize the precise state game designers want the user to be in for an optimal game experience. The work in [Nacke and Lindley 2008] uses a number of sensors in an effort to measure flow as bio-electric responses during game-play. Work by [Sweetser and Wyeth 2005] creates a player experience model designed to measure a user's flow during game-play. There is also a large body of work on why people enjoy computer games [Przybylski et al. 2010; Jansz and Tanis 2007; Yee 2006; Charles et al. 2005]

Game Level Optimization. We refer the readers to a recent survey in the area of Search-Based Procedural Content Generation (SBPCG) [Togelius et al. 2011] and provide a brief description below. The work in [Cardamone et al. 2011; Sorenson and Pasquier 2010b] uses evolutionary approaches for procedural level creation and the placement of game level design elements. None of these methods optimize the individual parameters that govern the control and movement of non-player characters in the level. Other complementary work [Browne and Maire 2010] enables authors to generate and evaluate new types of board games. Similar work in [Sorenson and Pasquier 2010a] optimizes platformer games to maximize "fun". In the area of parameter optimization, Berseth *et al.* [2014] proposes a new method to improve the behaviour of crowd simulations using combinations of objectives. Last, Togelius *et al.* [2013] uses multi-objective optimization for RTS game levels in order to generate a better game-play experience as a combination of objectives.

Comparison to Prior Work. Our work is similar to [Shi and Crawfis 2013] which uses optimization to find interesting variations in the game level, as well as [Bauer et al. 2013] which uses optimization to find new play space configurations for platformer games. However, we formulate a parameterization of the game level and evaluate each game level's expected difficulty. Additionally, the parameter space for optimization includes the level layout as well as parameters that control the behaviour and movement of non-player characters in the game, which in turn influence the overall level difficulty.

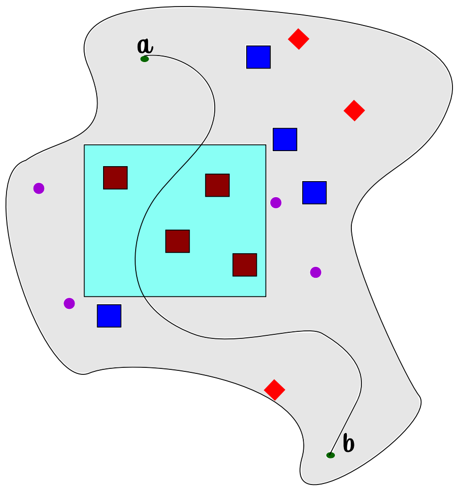


Figure 2: *Example scenario S. The diamonds are enemy NPCs,* ***a*** *is the starting position of the agent and* ***b*** *is the target location. The rectangles are static obstacles. The larger light rectangle that contains other rectangles is an optimize-area and the darker rectangles inside the optimize-area are parameterized obstacles.*

3 Framework

We introduce a parametrized representation of a game level which describes its layout and the initial configuration of the player and NPCs. By aggregating the expected interactions between a player and NPCs over the player's trajectory, we can compute a given level's expected difficulty.

3.1 Parametrized Game Levels

A scenario (game level) $S = \langle \mathbf{u}, \mathbf{A}, \mathbf{E} \rangle$ is a closed game environment which consists of a set of NPCs $\mathbf{A}$, a set of environment obstacles $\mathbf{E}$ and a player $\mathbf{u}$. Each NPC, $\mathbf{a} \in \mathbf{A}$, is defined by the tuple $\langle \mathbf{x}_a, \mathbf{G}_a, s_a, r_a, d_a, f_a \rangle$ where $\mathbf{x}_a$ is its initial position, $\mathbf{G}_a$ is a sequence of way-points along its desired path of travel, s_a is the agents speed, r_a is the range of attack, d_a is the agent attack damage and f_a defines the agent's attack accuracy. Each obstacle $\mathbf{e} \in \mathbf{E}$ is defined by its position and orientation, $\langle \mathbf{x}_e, \mathbf{q}_e \rangle$.

A player $\mathbf{u}$ is characterized by the tuple $\langle \mathbf{x}_u, \mathbf{q}_u, s_u, \mathbf{g}_u \rangle$ where $\mathbf{x}_u$ is the player's spawn position, $\mathbf{q}_u$ is the player's initial orientation, s_u is the player's movement speed, and $\mathbf{g}_u$ is the player's desired target (goal location). This provides an abstract representation of the player which can be used to estimate its expected movement in the game level. More accurate player models (e.g., recorded player input and movement trajectories) can easily be used instead, if appropriate. Figure 2 illustrates a simple game level where a player starts at a and has a final target at b. A parametrized version of the game level $S_{\mathbf{p}} = \langle \mathbf{u}, \mathbf{A}, \mathbf{E}, \mathbf{p} \rangle$ introduces a set of parameters $\mathbf{p}$ in the parameter space $\mathbf{P}$ on NPC, Player, or Obstacle configurations, NPC or Player parameter settings, and other Scenario parameters to procedurally generate new level variations.

3.2 Static Analysis of Game Levels

The main goal of this work is to be able to estimate the expected interactions between the player and the NPCs in the level without running expensive dynamic simulations. For this reason, we statically analyze game levels by computing expected paths of the player and agents based on the current environment configuration (scenario). For a given parametrized game level $S_{\mathbf{p}}$, the result of the static analysis is $\Phi(S_{\mathbf{p}}) = \langle \pi_{\mathbf{u}}, \{\pi_{\mathbf{a}} | \mathbf{a} \in \mathbf{A}\}, \mathbf{E} \rangle$ where $\pi_{\mathbf{u}}$ is the expected path of the player, and $\pi_{\mathbf{a}}$ is the expected path of agent $\mathbf{a}$ in the scenario.

A grid-based discretization of the game level proves to be insufficient for this analysis, where the result of the analysis would substantially differ based on granularity of discretization. Hence, we use a polygonal representation of traversable space in the game level which is used to compute collision-free paths. We chose to use Recast[1], a popular method for constructing navigation meshes for any environment geometry. After the navmesh is constructed, paths for the player and the agents are generated using traditional A* and line of sight smoothing. The computed paths do not make any assumptions about the behaviour or intentions of the agents in the game, and can be easily replaced by more specific pathing solutions, if desired. Similarly, the player's path can also be replaced with actual data available from game simulations. Player models similar to the work done in [Tomai et al. 2013] are not used in this system because optimization can not handle the random noise in this method.

3.3 Game Level Difficulty

Given the static analysis $\Phi(S_{\mathbf{p}})$ of a game level, we can estimate if the player $\mathbf{u}$ and agent $\mathbf{a}$ are interacting at some time t using:

$$I_{S_{\mathbf{p}}}(\mathbf{u}, \mathbf{a}, t) = \text{interact}(\pi_{\mathbf{u}}(t), \pi_{\mathbf{a}}(t)), \tag{1}$$

where

$$\text{interact}(\pi_{\mathbf{u}}(t), \pi_{\mathbf{a}}(t)) = \begin{cases} 1 & \text{dist}(\pi_{\mathbf{u}}(t), \pi_{\mathbf{a}}(t)) \leq r_a, \\ 0 & \text{otherwise.} \end{cases}$$

The expected difficulty for the player with respect to an agent $\mathbf{a}$ at a particular time t is given by:

$$d_{S_{\mathbf{p}}}(\mathbf{u}, \mathbf{a}, t) = (1 - \frac{\text{dist}(\pi_{\mathbf{u}}(t) - \pi_{\mathbf{a}}(t))}{r_a}) d_a f_a I_{S_{\mathbf{p}}}(\mathbf{u}, \mathbf{a}, t). \tag{2}$$

The cumulative difficulty for all agents $\mathbf{A}$ at a particular time t is:

$$d_{S_{\mathbf{p}}}(\mathbf{u}, t) = (1.1)^{|\mathbf{A}|} \sum_{\mathbf{a} \in \mathbf{A}} d_{S_{\mathbf{p}}}(\mathbf{u}, \mathbf{a}, t), \tag{3}$$

where d_a is the agent's attack damage, and f_a is the agent's attack accuracy. The multiplier $(1.1)^{|\mathbf{A}|}$ that is dependent on the cardinality of $\mathbf{A}$ is used to model the fact that difficulty increases with the number of simultaneously attacking agents. We wanted a particular trend of the curve when increasing the number of agents and this was found to be most suitable in comparison to other functions. The total expected difficulty in the time interval (ta, t_b) is computed as the line integral along the player's path.

$$d_{S_{\mathbf{p}}}(\mathbf{u}, t_a, t_b) = \int_{ta}^{t_b} d_{S_{\mathbf{p}}}(\mathbf{u}, t) dt, \tag{4}$$

where the net expected difficulty for the entire duration of the level is $d_{S_{\mathbf{p}}}(\mathbf{u}, 0, T)$ and T is the time it will take for the player to reach his/her target[2].

[1] https://github.com/memononen/recastnavigation

[2] $T = \frac{length(\pi_{\mathbf{u}})}{s_u}$

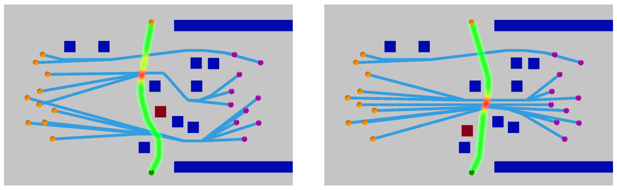


Figure 3: *Two example scenarios with only a small obstacle location change. The bottom scenario has moved one of the obstacle slightly down and to the left, putting it out of the path between player and the player's final goal. This change results in the scenario being almost twice as difficult as it was before.*

3.4 Difficulty Annotations

To visually describe, or annotate, the expected difficulty over time we use a normalized time-dependent difficulty path. This manifests as a heatmap like visualization along the player path. By normalizing the damage along the path, providing the user with a gradient legend, and displaying the total expected difficulty, we provide the means for intra-scenario comparisons of difficulty along computed paths. This path rendering allows the user to understand how difficulty progresses through time and to quickly make informed design decisions.

3.5 Authoring a Scenario

As a user-in-the-loop framework, a number of workflows are possible for authoring a scenario. Many elements may be added or edited to form a scenario. A basic workflow is described in Figure 4 as a realistic use case when generating an optimal scenario configuration based on a user-defined difficulty objective.

First, static level geometry is defined. This may involve importing entire game levels, typically from XML-based scene files, or working directly with a scene editor. The user, in this case, makes the decision to optimize an area of the level such that obstacles are given configurations that affect not only the overall flow of the NPCs and player but the expected difficulty. Using the annotations on expected difficulty, the user can make decisions on new configurations.

Another use of the framework may be to generate a number of desired configurations, which are later selected by some algorithm. For example, level layout may change based on player performance metrics. In this way, the final authoring of the level happens at run time.

4 Optimization Formulation

In this section we outline the construction of an optimization objective. This includes the use of boundary constraints to enforce desired conditions and penalty methods to discard poorly structured scenarios.

4.1 Boundary Constraints

We use boundary constraints to force obstacles and agents to be within an area determined by the level designer. Boundary constraints can be enforced in a number of ways. We choose to clamp samples to be within a designated threshold and then add substantial penalty with respect to the distance the sample is from the feasible region. This avoids using values that might cause issues with the

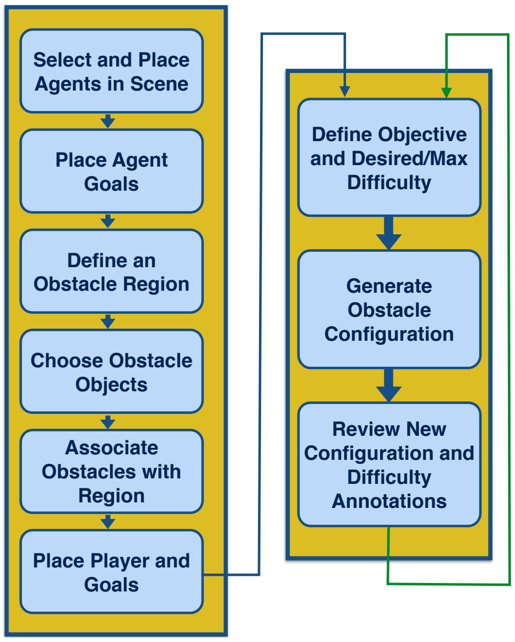

Select and Place Agents in Scene
Place Agent Goals
Define an Obstacle Region
Choose Obstacle Objects
Associate Obstacles with Region
Place Player and Goals
Define Objective and Desired/Max Difficulty
Generate Obstacle Configuration
Review New Configuration and Difficulty Annotations

Figure 4: *Example workflow of a basic use of the framework. In the left box, the user defines elements forming a parametrized scenario S_p. In the right box, the user iterates on optimization results converging to a desired configuration $\mathbf{p}'$.*

stability of the system and best compliments our penalty method described in Section 4.2.

In Figure 5 we can see how obstacle locations can be constrained, as illustrated by the bounding rectangle called an optimize-area which contains the 9 obstacles being optimized. Each optimize-area has a position and scale that is used to determine its bounds. As well, the settings for the NPCs can be bounded by associating them with an optimize-area. In this fashion NPCs can be grouped together during optimization where each group can have different parameter settings and bounds. With the introduction of these constraints ($\mathbf{C}$) the definition for a parameterized scenario $S_\mathbf{p}$ becomes $\langle \mathbf{u}, \mathbf{A}, \mathbf{E}, \mathbf{p}, \mathbf{C} \rangle$.

4.2 Penalty Method

We employ an additional penalty constraint to keep the parameterized obstacles from overlapping. We can compute something close to the max difficulty (d_{max}) for a scenario and add, or subtract depending on the objective, the following penalty for every pair of overlapping obstacles:

$$p_{S_\mathbf{p}}(\mathbf{E}) = \sum_{\mathbf{e}_1 \in \mathbf{E}} \sum_{\mathbf{e}_2 \in \mathbf{E}} \begin{cases} 0 & \mathbf{e}_1 = \mathbf{e}_2 \\ \text{overlap}(\mathbf{e}_1, \mathbf{e}_2) d_{max}. & \text{otherwise.} \end{cases} \quad (5)$$

4.3 Optimization Objective

Under proper constraints we can make parts of a scenario modifiable. In Figure 2 a number of these items can be seen. The parts of the scenario we can modify include: (1) obstacle locations, (2) damage effect radius r_a, agent speed s_a, agent attack damage d_a, and agent accuracy f_a. Given a set of parameters $\mathbf{p}$, we can maximize the expected difficulty of a scenario $S_\mathbf{p}$ as follows:

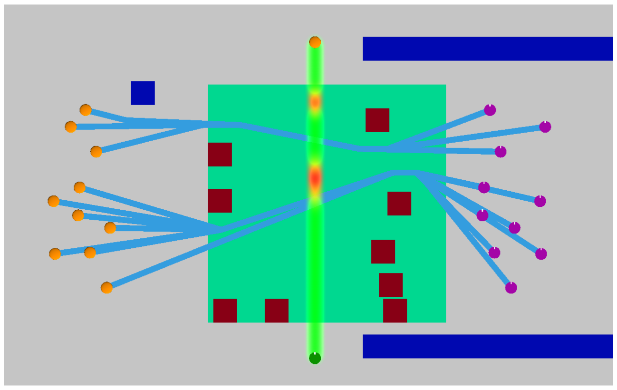


Figure 5: *The highlighted rectangle shows how the bounds for obstacle location optimization can be defined. The location and scale of the rectangle can be easily changed.*

$$\mathbf{p}' = \arg\max_{\mathbf{p} \in \mathbf{P}} |d_{S_\mathbf{p}}(\mathbf{u}, 0, T) - p_{S_\mathbf{p}}(\mathbf{E})|. \quad (6)$$

The optimized scenario is $S' = \langle \mathbf{u}, \mathbf{A}, \mathbf{E}, \mathbf{p}', \mathbf{C} \rangle$.

Desired Difficulty. It is not typically the goal of a level designer to make the level as hard as possible. For a designer that wants a particular difficulty we can easily modify the difficulty metric and use the magnitude of the difference between the calculated difficulty and the desired difficulty (d_{des}) as follows:

$$d_{S_\mathbf{p}}(\mathbf{u}, 0, T, d_{des}) = |d_{S_\mathbf{p}}(\mathbf{u}, 0, T) - d_{des}|. \quad (7)$$

Optimizing with this metric gives the author more control over the difficulty for the generated scenario. To produce the most difficult scenario one would only need to set the difficulty d_{des} to an extremely high value and multiply the penalty method described in Section 4.2 by -1. In order for this method to work well we must ensure $d_{max} >> d_{des}$.

Optimization Method. The difficulty function can have large discontinuities. These discontinuities occur when nodes in a path are exchanged for different nodes after obstacles are moved in the environment. Given that our problem has discontinuities and is in the space of non-linear and non-convex functions there are few optimization algorithms that can manage this type of problem. From these options we choose to use the Covariance Matrix Adaptation Evolution Strategy technique (CMA-ES) [Hansen and Ostermeier 1996]. The CMA algorithm is well suited to this domain for many reasons: it is straightforward to implement, it can handle ill-conditioned objectives and noise and it is very competitive in converging to an optimal value in few iterations.

5 Benchmarks and Experiments

We perform experiments that include slightly altering scenarios to show the significant change in difficulty. Also, we show how our framework performs, optimizing for minimum difficulty and for a desired difficulty. The first benchmark scenario is a simple warehouse layout with a number of patrolling agents and a few stationary enemies. In the second scenario, the player needs to cross a street while being attacked by zombies traveling perpendicular to the player's path. The layout of the last scenario is from the computer game Baldur's Gate II. Here we add components from the previously described experiments so as to make the scenario extra challenging. We use the Unity game engine to render all our results. The annotation of a scenario can be done in real time.

5.1 Warehouse Scenario

The first example scenario we evaluate is a basic warehouse with a number of patrolling agents and a few turrets. An example of this scenario can be seen in Figure 6.

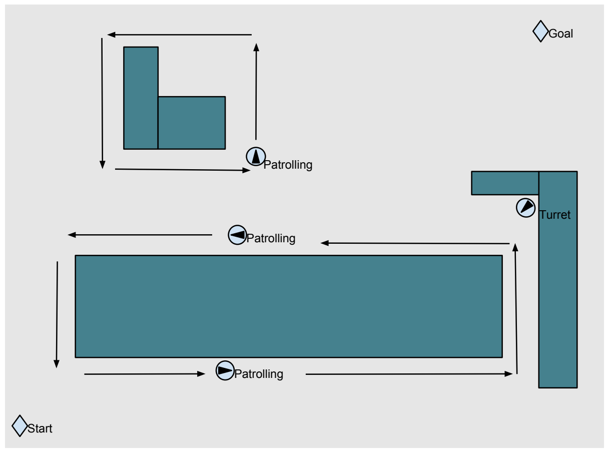


Figure 6: *An example scenario of a simple warehouse. This warehouse has a few patrolling agents and some stationary turrets.*

For this example, we can change the settings for each agent in the scenario to alter the resultant difficulty in the simulation. While it might seem overly simplistic to lower the expected difficulty of this scenario, the agent setting for speed makes the difficulty optimization non-trivial.

We start by annotating a warehouse-like scenario and slightly altering the configuration of the scenario. This small change results in a large difference in the difficulty value. This effect can be seen in Figure 7(a) and (b) where the path for the player has been annotated, where difficulty is expected to occur (in red).

Next we optimize the warehouse example for a number of desired difficulties. We bound the settings for the agents in the scenario with $\mathbf{C}$ and optimize for $d_{des} = 20$ and $= 50$. The min, max, default and optimal settings for $d_{des} = 20$ can be found in Table 1, while the associated result can be seen in Figure 7(c).

parameter	default	min	max	optimal
attack damage	2.30	1	5	2.00
attack radius	1.30	1	10	2.37
speed	1.33	1.00	1.50	1.25
accuracy	0.87	0.20	0.95	0.95

Table 1: *The default parameter settings for the Warehouse scenario and the lower and upper bounds for the optimization constraints* ($\mathbf{C}$)*. The last column are the optimal settings for difficulty* $d_{des} = 20$.

5.2 Zombie Scenario

This scenario is created to showcase a scene common in zombie hoard computer games. The scenario shows a single player that has to cross a street littered with obstacles. As well, there are a number of enemy zombies travelling perpendicular to the player as can be seen in Figure 8.

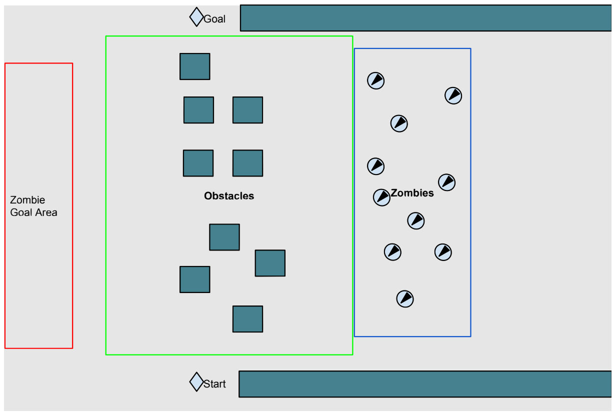


Figure 8: *A street crossing scenario. When the player crosses the street he/she is attacked by zombies coming from the right.*

The parameters of this scenario are the locations of the obstacles in between the player and its target. The setting for the enemy zombies remains fixed.

In this scenario we showcase the framework's ability to optimize the locations of obstacles in the game level. Similar to the warehouse example, we start by showing the effect of slightly modifying the configuration of the scenario and observing the result of this modification, see Figure 9(a) and (b). In this example, significant changes in difficulty can be observed. The location difficulty is expected along the player's path can change significantly.

The results in Figure 9(c) and (d) show how easy it is to optimize for a particular difficulty. It is also interesting to note that there is a significant number of configurations that have very similar difficulties which can be generated by using the framework repeatedly for the same d_{des}, are show in Figure 10(a)-(d).

5.3 Baldur's Gate II Scenario

This scenario is from the movingAI benchmarks [Sturtevant 2012] and is originally from the Baldur's Gate II video game. Here, we modify it to include many of the elements from the previous two examples. There is a large group of enemies travelling somewhat perpendicular to the player's path. There are a number of patrolling agents that will be difficult for the player to avoid.

The scenario elements that are modified by the optimization include the settings for different groups of agents, the locations of the two separate groups of obstacles, and the settings for the patrolling agents.

In this last example we combine many of the elements from previous examples. There are two optimizing areas in this example with many obstacles of different sizes in each. There are also different kinds of agents, one large set of enemies sweeping across the scenario from top to bottom, one set of patrolling agents in the middle of the level and another closer to the player's goal. Two variations of this scenario can be seen in figures 12(a) and (b).

In figures 12(c) and (d) the difference in difficulty can be clearly seen from the annotated player's path. The first example in (c) is optimized for $d_{des} = 200$ and for $d_{des} = 1200$ in (d). Even though the player paths are similar, changes in the scenario affect the agents. The affected agents in turn significantly affect the expected difficulty of the scenario.

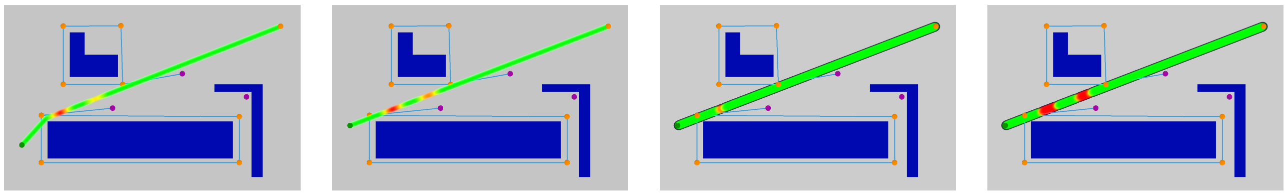

(a) (b) (c) $d_{des} = 20$ (d) $d_{des} = 50$

Figure 7: *Scenarios (a) and (b) show the effect on expected difficulty of slightly altering the initial position of the player. Scenarios (c) and (d) are the result of optimizing for desired difficulties* 20 *and* 50*.*

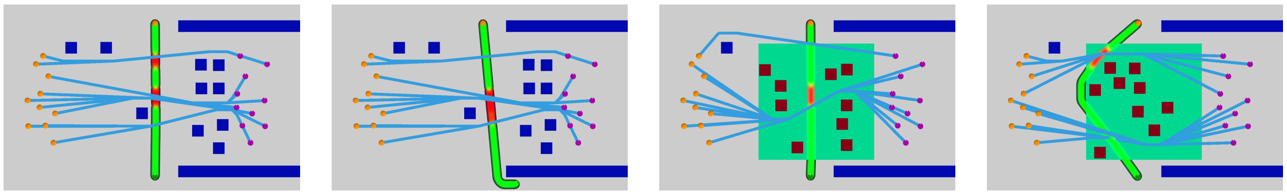

(a) (b) (c) $d_{des} = 20$ (d) $d_{des} = 50$

Figure 9: *Figure (a) is the default scenario. Figure (b) shows the effect of slightly modifying the starting position of the player. Figures (c) and (d) show the result of optimizing for specific desired difficulties.*

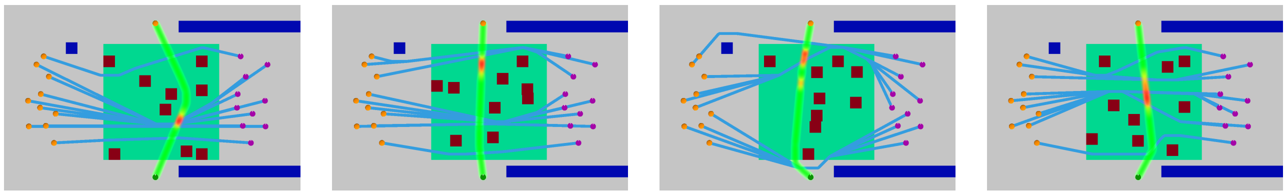

(a) (b) (c) (d)

Figure 10: *Potential scenario variations for the same desired difficulty* $d_{des} = 20$*.*

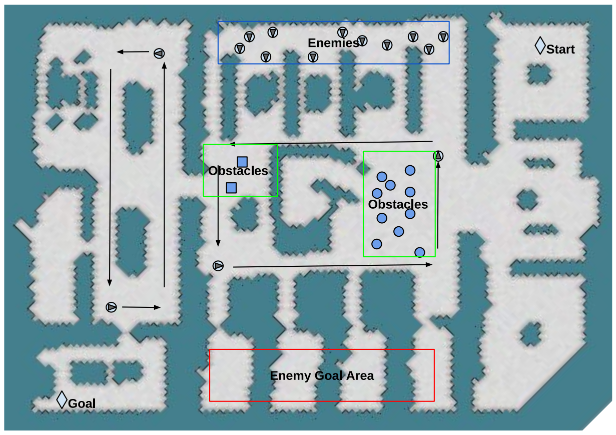


Figure 11: *The basic layout for this scenario was taken from a game level in Baldur's Gate II. We added a number of elements to make it more complex and challenging. These elements include a large groups of agents, and multiple optimization-areas which are associated with multiple obstacles and agents.*

5.4 Performance

The running time for our framework is dependent on the number of polygons in the scenario as well as the number of items that are being optimized. Optimizations are performed on a quad-core 2.4ghz computer and optimizing the zombie example for 200 iterations with a population of 20 takes less than a minute. The Baldur's Gate II example can take much longer because of the number of polygons in the scenario.

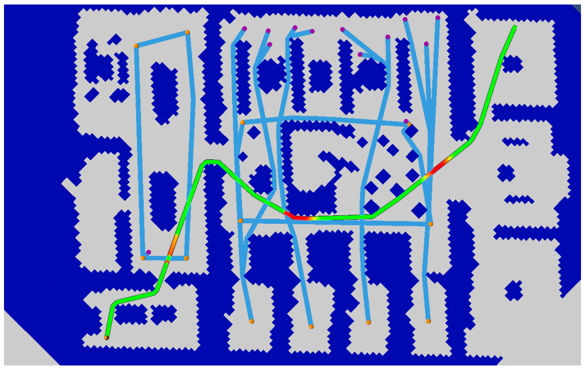
It is important to note that scenarios need to be sufficiently difficult for the optimization to work well. If the scenario is simple it could be trivial to get a difficulty of 0. This will cause the optimization to terminate quickly with a simple result.

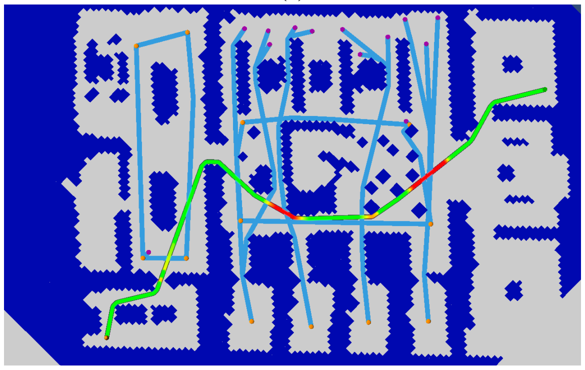
Given the dimensionality of the problem, when working with large and complex scenarios the population size should be increased. This will increase the chances of the system finding the global optimum. The system is initialized with the user provided initial scenario. This initial configuration should be a best guess at what is the optimal solution ($\mathbf{p}'$) to start the optimization.

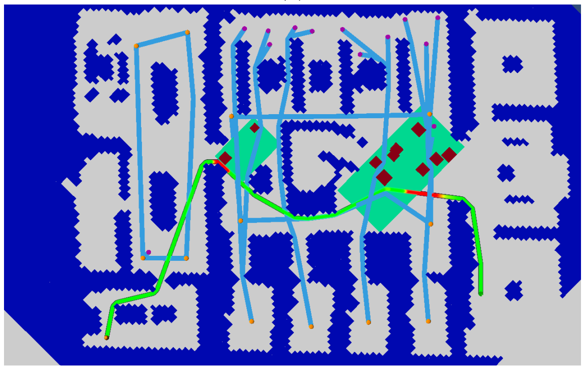
6 Conclusion

In this paper we have explored the possibility of characterizing the difficulty of a game level by predicting the expected interactions between players and non-player characters. Based on this notion of difficulty, we have developed an optimization framework for automatically synthesizing level variations that meet author constraints while optimizing for a desired level of difficulty. Our framework allows level designers to interactively generate potentially multiple variations of a game's level that satisfy given constraints on layout while maintaining a desired level of difficulty.

Our difficulty measure is general, robust, and sensitive to small


(a)


(b)


(c) $d_{des} = 200$

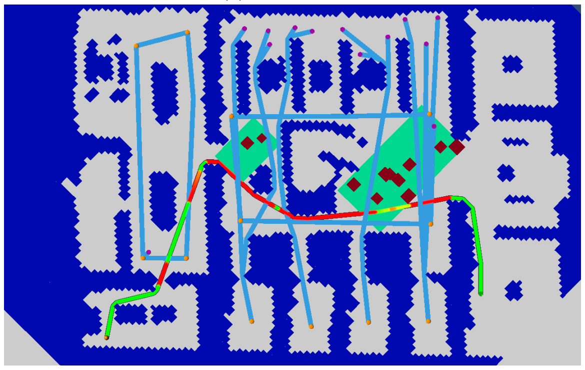

(d) $d_{des} = 1200$

Figure 12: *Figures (a) and (b) illustrate the change in the amount and location of expected difficulty by altering the initial location of the player. In Figures (c) and (d), the parameterized scenario was optimized for* $d_{des} = 200$ *and* $= 1200$.

changes in the level definition which may drastically impact how the level plays out. We demonstrate our framework on different game benchmarks from popular games by computing their difficulty scores as well as proposing minor variations that significantly improve the players experience, by either making them more challenging, or reducing the difficulty for an easier playthrough.

Assumptions for NPC Behaviour. Defining what behaviour the NPCs should exhibit is a difficult task, and is also specific to the game genre and the type of AI techniques used. For instance, our current NPC model does not account for the fact that an agent might follow the player once the agent sees the player. Such a behaviour would require a formulation of "interaction persistence", which in turn would require modelling the player's ability to evade enemies. Our current implementation models NPC behaviour as an author-specified sequence of waypoints which can be a cycle to define a patrolling behaviour. This simple model can be easily replaced with game-specific logic that governs NPC AI, if appropriate.

Limitations and Future Work. The current penalty function used in the optimization can be made smoother by calculating the amount of overlapping area between obstacles. The player and NPC models could be enhanced to capture more complex behaviour patterns and game-specific AI. The current navigation mesh needs to be recomputed for each new environment configuration. Repairing strategies can be employed to refine the navigation mesh and avoid expensive re-computation within the optimization loop. In the future we would like to explore sampling methods over possible player paths. Dealing with cooperative and competitive multiplayer scenarios is also a subject for future exploration. We could also perform dynamic analysis where we execute agent simulations using a steering algorithm such as [Singh et al. 2011a; Kapadia et al. 2009; Helbing et al. 2000] to analyze difficulty. We could also use more complex agent models for example [Singh et al. 2011b] for simulation. We could also explore the enforcement of group constraints, using methods similar to the work done in [Schuerman et al. 2010], in the enemy agents when running the analysis. Lastly, this work could benefit from the collection of user play data, to compare the subjective difficulty of scenarios.

References

BAUER, A., AND POPOVIĆ, Z. 2012. Rrt-based game level analysis, visualization, and visual refinement. In *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*.

BAUER, A. W., COOPER, S., AND POPOVIC, Z. 2013. Automated redesign of local playspace properties. In *FDG*, Citeseer, 190–197.

BERSETH, G., KAPADIA, M., AND FALOUTSOS, P. 2013. Steerplex: Estimating scenario complexity for simulated crowds. In *Proceedings of Motion on Games*, ACM, New York, NY, USA, MIG '13, 45:67–45:76.

BERSETH, G., KAPADIA, M., HAWORTH, B., AND FALOUTSOS, P. 2014. SteerFit: Automated Parameter Fitting for Steering Algorithms. Eurographics Association, Copenhagen, Denmark, V. Koltun and E. Sifakis, Eds., 113–122.

BROWNE, C., AND MAIRE, F. 2010. Evolutionary game design. *Computational Intelligence and AI in Games, IEEE Transactions on 2*, 1 (March), 1–16.

CARDAMONE, L., YANNAKAKIS, G. N., TOGELIUS, J., AND LANZI, P. L. 2011. Evolving interesting maps for a first person shooter. In *Applications of Evolutionary Computation*. Springer, 63–72.

CHARLES, D., KERR, A., MCNEILL, M., MCALISTER, M., BLACK, M., KCKLICH, J., MOORE, A., AND STRINGER, K. 2005. Player-centred game design: Player modelling and adaptive digital games. In *Proceedings of the Digital Games Research Conference*, vol. 285. 00100.

CSIKSZENTMIHALYI, M. 1990. *Flow: The psychology of optimal experience*. Perennial Modern Classics. Harper & Row.

GÜTTLER, C., AND JOHANSSON, T. D. 2003. Spatial principles of level-design in multi-player first-person shooters. In *Proceedings of the 2Nd Workshop on Network and System Support for Games*, ACM, New York, NY, USA, NetGames '03, 158–170.

HANSEN, N., AND OSTERMEIER, A. 1996. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *IEEE International Conference on Evolutionary Computation*, 312–317.

HELBING, D., FARKAS, I., AND VICSEK, T. 2000. Simulating dynamical features of escape panic. *Nature 407*, 6803, 487–490.

HORSWILL, I. D., AND FOGED, L. 2012. Fast procedural level population with playability constraints. In *AIIDE*.

JANSZ, J., AND TANIS, M. 2007. Appeal of playing online first person shooter games. *CyberPsychology & Behavior 10*, 1, 133–136.

KAPADIA, M., SINGH, S., HEWLETT, W., AND FALOUTSOS, P. 2009. Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '09, 215–223.

LIAPIS, A., YANNAKAKIS, G. N., AND TOGELIUS, J. 2013. Towards a generic method of evaluating game levels. In *Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference*.

NACKE, L., AND LINDLEY, C. A. 2008. Flow and immersion in first-person shooters: Measuring the player's gameplay experience. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, ACM, New York, NY, USA, Future Play '08, 81–88.

PRZYBYLSKI, A. K., RIGBY, C. S., AND RYAN, R. M. 2010. A motivational model of video game engagement. *Review of General Psychology 14*, 2, 154.

SCHUERMAN, M., SINGH, S., KAPADIA, M., AND FALOUTSOS, P. 2010. Situation agents: agent-based externalized steering logic. *Comput. Animat. Virtual Worlds 21* (May), 267–276.

SHI, Y., AND CRAWFIS, R. 2013. Optimal cover placement against static enemy positions. In *FDG*, 109–116.

SINGH, S., KAPADIA, M., HEWLETT, B., REINMAN, G., AND FALOUTSOS, P. 2011. A modular framework for adaptive agent-based steering. In *ACM I3D*, 141–150.

SINGH, S., KAPADIA, M., REINMAN, G., AND FALOUTSOS, P. 2011. Footstep navigation for dynamic crowds. *Computer Animation and Virtual Worlds 22*, 2-3, 151–158.

SMITH, A., AND MATEAS, M. 2010. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, 273–280.

SMITH, A. M., ANDERSEN, E., MATEAS, M., AND POPOVIĆ, Z. 2012. A case study of expressively constrainable level design automation tools for a puzzle game. In *Proceedings of the International Conference on the Foundations of Digital Games*, ACM, New York, NY, USA, FDG '12, 156–163.

SORENSON, N., AND PASQUIER, P. 2010. The evolution of fun: Automatic level design through challenge modeling. In *Proceedings of the First International Conference on Computational Creativity (ICCCX). Lisbon, Portugal: ACM*, 258–267.

SORENSON, N., AND PASQUIER, P. 2010. Towards a generic framework for automated video game level creation. In *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation - Volume Part I*, Springer-Verlag, Berlin, Heidelberg, EvoApplicatons'10, 131–140.

STURTEVANT, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games 4*, 2, 144 – 148.

SWEETSER, P., AND WYETH, P. 2005. Gameflow: A model for evaluating player enjoyment in games. *Comput. Entertain. 3*, 3 (July), 3–3.

TOGELIUS, J., YANNAKAKIS, G. N., STANLEY, K. O., AND BROWNE, C. 2011. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on 3*, 3, 172–186.

TOGELIUS, J., PREUSS, M., BEUME, N., WESSING, S., HAGELBÄCK, J., YANNAKAKIS, G. N., AND GRAPPIOLO, C. 2013. Controllable procedural map generation via multiobjective evolution. *Genetic Programming and Evolvable Machines 14*, 2 (June), 245–277.

TOMAI, E., SALAZAR, R., AND FLORES, R. 2013. Mimicking humanlike movement in open world games with path-relative recursive splines. In *AIIDE*.

YEE, N. 2006. Motivations for play in online games. *CyberPsychology & behavior 9*, 6, 772–775.