Automatic Puzzle Level Generation: A General Approach using a Description Language

Ahmed Khalifa and **Magda Fayek**
Computer Engineering Department
Faculty of Engineering, Cairo University
Cairo University Road, Giza, Egypt
amidos2002@hotmail.com, magdafayek@ieee.org

Abstract

In this paper, we present a general technique to evaluate and generate puzzle levels made by *Puzzle Script*, a videogame description language for scripting puzzle games, which was created by Stephen Lavelle(**?**). In this work, we propose a system to help in generating levels for Puzzle Script. Levels are generated without any restriction on the rules. Two different approaches are used with a trade off between speed (Constructive approach) and playability (Genetic approach). These two approaches use a level evaluator that calculates the scores of the generated levels based on their playability and challenge. The generated levels are assessed by human players statistically, and the results show that the constructive approach is capable of generating playable levels up to 90%, while genetic approach can reach up to 100%. The results also show a high correlation between the system scores and the human scores.

Introduction

During the early days of Video Games, games were created by few people in their spare time. Most of the time was spent in programming the game, while a small portion was dedicated for graphics, sounds, and music because of the technical limitations of the devices at that time. As these limitation are no more, producing a game takes more time than before. Most of that time is spent on creating content for the game (graphics, music, sounds, levels, and ...etc)(**?**); for example creating graphics for a huge main stream game may take hundreds of artist working for a year or two. That is why the production cost of a huge game reaches millions of dollars(**?**).

That huge production cost is one of the reasons for the use of Procedural Content Generation(PCG). PCG means generating game content using a computer. It was first developed due to technical limitations (small disk space) and the need to provide a huge amount of content(**?**). Although technical difficulties become history and storage is no longer a problem, PCG is still one of the hot topics in Video Games Industry and Research. PCG helps us reduce development time and cost, be creative, and understand the process of creating game content. PCG can be used to generate different game aspects for example Textures, Sounds, Music, Levels, Rules, and ...etc.

Level Generation has been in industry since dawn of games in order to decrease the game size, but it is now used to introduce huge amount of levels that humans can not generate manually in reasonable time. Level Generation has always been done for a specific game using lots of hacks to improve the output result. These hacks cause the output levels to follow certain guidelines which may cause elimination of huge amounts of possible levels but on the other hand these guidelines ensure that the output levels are all playable (can reach goal of the game) and satisfactory by all players(**?**).

In this paper, we propose a system to generate playable puzzle levels without any restrictions or hacks. we utilize small prior knowledge about puzzle script to ensure playability and challenge.

Background

We can not generate general levels without having a methodology to describe the games. Video Game Description Language(VGDL) was originally invented to help on the work for General Video Game Playing(GVGP)(**?**) at Standford University. Puzzle Script(PS) is a VGDL created by Stephan Lavelle to help game designers and developers to create puzzle games(**?**). Games generated by PS are time stepped games similar to Sokoban(**?**). PS file starts with some meta data like game name, author name, and website then it is divided into 7 sections objects, legend, sounds, collision layers, rules, win conditions, and levels.

In this work, we focus on rules, win conditions, and levels section. Rules are a set of production rules that govern how the game will be played. For example, **[> Player | Crate] -> [> Player | > Crate]** means if there is a Player and Crate beside each other, and the Player moves towards the Crate, then both the Player and the Crate will move in the same direction. Win conditions are group of rules that identify when the level should end. Levels are 2D matrices showing the current configuration for each game level using objects identified in objects section.

Literature Review

As there is nothing before like this work, this section will show all previous work that can be slightly related to our

problem. One of the earliest research in Puzzle Games was by Murase et al.(**?**). Murase et al. work focused on generating well designed solvable levels for Sokoban(**?**). Breadth First Search(BFS) is used to check playability. Results show that for every 500 generated levels only 14 are considered as good levels. These levels are characterized by having a short solution sequence. Taylor and Parberry(**?**) followed Yoshio Murase et al.(**?**) work to improve generated level quality. Their system places the crates at the farthest possible location using a similar algorithm to BFS. The generated levels are not suffering from the problem of short solution sequences presented in Yoshio Murase et al.(**?**) work.

Rychnovsky(**?**) work focused on generating levels for his new game Fruit Dating(**?**). Rychnovsky developed a level editor that can be used to generate new levels or test playability of certain level. Generating new levels is done by generating level layout and placing game objects based on some prior game knowledge, then check for a solution using a similar algorithm to BFS. The technique doesn't take more than couple of minutes to generate a level with no direct way to influence difficulty in the generated levels.

Shaker et al.(**?**) worked on generating levels for physics based puzzle games. They applied their technique on Cut The Rope(CTR)(**?**). Shaker et al. used Grammar Evolution (GE) technique to generate levels for CTR. The grammar is designed to ensure that every game object appears at least one time. The fitness function depends on some heuristic measures based on prior knowledge about the game and the result of several playouts using a random player. Shaker et al. generated 100 playable levels and analyzed them according to some metrics such as frequency, density, and ...etc. Shaker et al.(**?**) conducted their research on CTR to improve generated level quality. They replaced the random player with an intelligent one. The generated levels are far more diverse because the random player discards some potential levels in the search space.

Shaker et al.(**?**) introduced a new generation technique named Progressive Approach. It can be used on any kind of games to reduce the generation time. Progressive Approach starts by using GE to generate a time-line of game events, then an intelligent player is used to map and evaluates the time-line to a playable level. Shaker et al. tested the new technique on CTR and compared its results with their previous work(**?**). The results indicates a huge decrease in generation time, but the quality of the levels depends on the intelligent player used in the mapping process.

Smith et al.(**?**) worked on generating puzzle levels for Refraction(**?**). The system starts by generating a solution outline, then it translates into a geometric layout, at last the system tests the generated level for playability. Smith et al. implemented the system into two different ways (Algorithmic approach and Answer Set Programming (ASP)). Results shows that ASP is faster than Algorithmic approach, while Algorithmic approach produces more diverse levels than ASP.

Methodology

Level Generation is not an easy task specially when the game rules are not known before generation. Although some of the previous research suggested a general technique to generate levels, it is still based on designing a game specific fitness function. Our approach relies heavily on the understanding of the current game rules and some prior knowledge about Puzzle Script language. Figure 1 shows a high level block diagram of the system. The following subsections will describe each block in details.

The system starts by analyzing the current game rules using a Rule Analyzer. The output of the Rule Analyzer (Rule Analysis) and the Level Outlines are fed to a Level Generator. The Level Generator generates some initial level layouts using Genetic Algorithm(GA) or Constructive Algorithm. The generated levels are subjected to a Level Evaluator. The Level Evaluator measures level playability and challenge using an automated player. The following subsections describe each of these components in details.

Rule Analyzer

The Rule Analyzer is the first module in our system. It analyzes game rules and extract some useful information about each object. The extracted information is fed to the Level Generator and the Level Evaluator. Each object is assigned:

- *Type:* Object type depends on its presence in the Puzzle Script file. There are 4 different types:
  - *Rule Object:* Any object that appears in a rule is defined as a rule object.
  - *Player Object:* It is defined by name "Player" in the Puzzle Script.
  - *Winning Object:* They are objects appearing in the winning condition.
  - *Solid Object:* All objects that does not appear in any rule but on the same collision layer with a Rule Object.
- *Subtype:* each Rule Object is assigned a Subtype based on its presence in game rules. These subtypes are:
  - *Critical Object:* is an object that has appeared with the Player object and one of the Winning Objects in the rules.
  - *Normal Object:* same like the Critical Object but it only appears with one of them.
  - *Useless Object:* is an object that neither appears with the Player Object nor the Winning Objects in any rule.
- *Priority:* It reflects the number of times each object appears in the rules.
- *Behaviors:* Behaviors are analyzed from the difference between the left hand side and the right hand side of each rule for every object. Every object can have one or more behavior. There are 4 kinds of behaviors:
  - *Move:* The object on the left hand side have different movement than the right hand side.

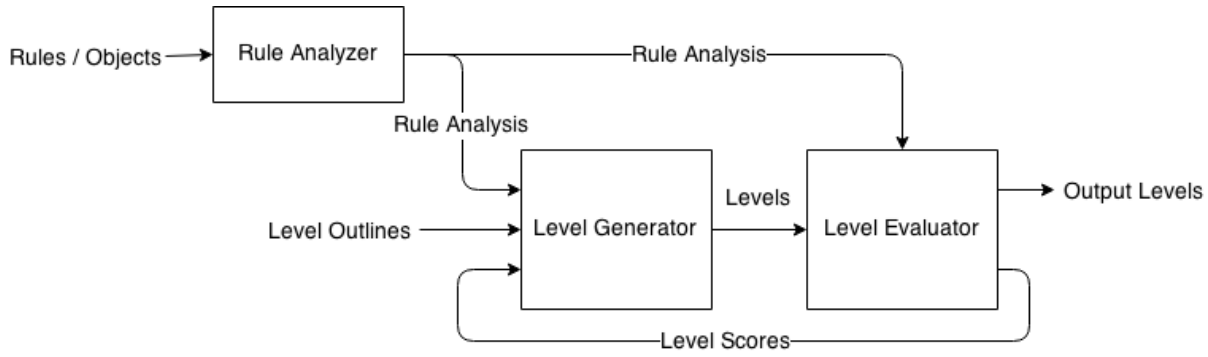

Figure 1: High level system block diagram for Level Generation

  - *Teleport:* The object on the left hand side have different location in the rule in the right hand side.
  - *Create:* The number of a certain object on the left hand side is less than its number on the right hand side.
  - *Destroy:* The number of a certain object on the left hand side is more than its number on the right hand side.
- *Minimum Number:* It is the maximum number of times for an object to appear in the left hand side of game rules except for objects with Create behavior. In create rules, the numbers reflects the least number of appearances for the object.
- *Relations:* It is a list of all objects that appears in the same rule with a certain object.

Level Generator

The Level Generator is responsible for creating a level in the best possible way. Two approaches were used to generate levels. The following subsections will discuss each one of them.

Constructive Approach Constructive Approach uses information from the Rule Analyzer to modify the Level Outlines. In this approach, several levels are generated using a certain algorithm and the best levels are selected. A pseudo code for the algorithm is presented in Algorithm 1.

The algorithm consists of two main part. The first part is responsible for determining the amount of objects that should be presented in the current level outline. Each object type contributes by a percentage equal to its minimum number to make sure that all rules can happen. Winning objects have an equal amount of objects except if any of these objects have a Create behavior. The second part is responsible for inserting game objects in the most suitable location. All the insertion algorithms need to find the most suitable empty locations to insert the new Object. The most suitable location is calculated based on the features of the inserted object. If the object has a Move behavior, it should be inserted at spots with the most free locations around it. Otherwise any random free location is okay. The second winning object is inserted on the same place of the first one if No rule is used.

Algorithm 1: Pseudo algorithm for the Constructive Approach

Data: level outline, rule analysis
Result: modified level outline

numberObjects = Get the number of objects for each object type;

levelOutline = Insert Solid Objects in the level outline;
levelOutline = Insert Winning Objects in the level outline;
levelOutline = Insert Player Object in the level outline;
levelOutline = Insert Critical Objects in the level outline;
levelOutline = Insert Rule Objects in the level outline;

return levelOutline;

All critical objects are inserted at least one time, while normal rule objects are selected based on their Priority feature.

Genetic Approach This method uses GA to evolve level outlines to playable levels. Elitism is used to ensure that the best levels are carried to the next generation.

Chromosome Representation: In this technique levels are represented as 2D matrix. Each location value represents all the objects at that location.

Genetic Operators: Crossover and Mutation are used to ensure better levels in the following generations. One point crossover is used where level rows are swapped between the two selected chromosomes. Mutation changes any random selected position using the following:

- *Creating an object:* a random object is selected to replace an empty position in the level.
- *Deleting an object:* a random object from the level is deleted.
- *Changing object position:* a random empty position is swapped with a non-empty one.

The mutation operation happens by subjecting the level outline to these 3 methods using different probabilities. The creating and deleting an object have lower probability than

the changing object position.

Initial Population: Three different techniques are used to generate an initial population for the GA. These techniques are:

- *Random Initialization:* The population is initialized as mutated versions of the empty level outline.
- *Constructive Initialization:* The population is initialized using the Constructive Approach algorithm.
- *Mixed Approach:* The population is initialized as a mixture between the Random Initialization, the Constructive Initialization, and mutated version of the Constructive Initialization.

Level Evaluator

Level Evaluator is responsible for evaluating the generated levels. Level evaluation is based on some global knowledge about the Puzzle Script Language and Puzzle Games. The evaluation takes place by measuring the level playability and some heuristics measures. Level playability is achieved by using an automated player which will be discussed later. These heuristics measures ensure that the solution have more moves with some thinking ahead. Figure 2 shows several levels designed for Sokoban game where all are playable but some of them are more interesting than the others.

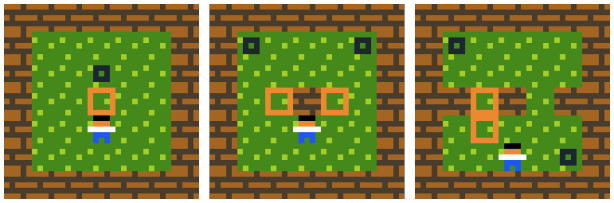

Figure 2: Examples on different levels for Sokoban

Automated Player Our Level Evaluator uses a modified version of the BestFS Algorithm as the automated player. BestFS Algorithm was introduced in Lim et al.(**?**) work. BestFS is similar to BFS algorithm but instead of exploring states sequentially, it sorts them according to a score generated from a fitness function. This causes the algorithm to explore the more important nodes first, helping it to reach the solution faster.

In any proper game, rules must be applied before achieving the winning condition. Based on that fact, we extended Lim et al. metrics to measure the distance between rule objects in the left hand side of each rule. For example, Figure 3 shows a level from a game called LavaGame. LavaGame is a puzzle game where the goal is to make the player reaches the exit. The path towards the exit is usually stuck by lava which can be destroyed by pushing a crate over it. Player's aim is to move crates towards the lava to unblock his path towards the exit. This aim is somehow explained in the game rules, so by using the output of Rule Analyzer, we can know which objects need to be closer.

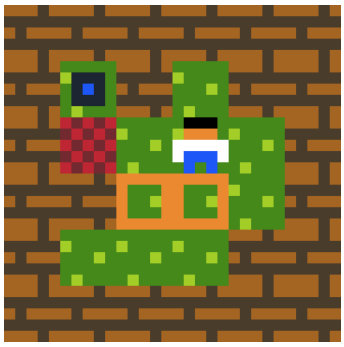

Figure 3: Example level from LavaGame showing the usage of the new metric

Heuristic Measures Heuristic measures are calculated using a weighted function of six measures.

- *Playing Score (P_{score}):* Playing score is used to ensure level playability. A float value is assigned for how much the level is near the solution. Based on the work by Nielsen et al.(**?**), a score is calculated for the initial level state using the same way and subtracted from previous value. The Playing Score can be expressed by the following equation:

$$P_{score} = S_{play} - S_{nothing}$$

  where S_{play} is the automated player score and $S_{nothing}$ is the initial level score.

- *Solution Length Score (L_{score}):* Bigger levels needs longer solution, a score is given to the ratio between solution length and level area. We analyzed 40 hand crafted levels with different area from 5 different games. A histogram is plotted for the ratio and shown in Figure 4. The histogram seems to follow a Normal Distribution with $\mu = 1.221$ and $\sigma = 0.461$. Based on that, the Solution Length Score is expressed by the following equation:

$$L_{score} = Normal(\frac{L}{A}, 1.221, 0.461)$$

  where $Normal(ratio, \mu, \sigma)$ is a normal distribution function, L is the solution length, and A is the level area.

- *Object Number Score (N_{score}):* The Object Number Score is calculated by the following equation:

$$N_{score} = 0.4 * N_{rule} + 0.3 * N_{player} + 0.3 * N_{winning}$$

  - *Number of Rule Objects (N_{rule}):* The number of times an object should appear in the level must be greater than or equal its minimum number property from the Rule Analyzer to ensure the possibility of applying all rules.
  - *Number of Players (N_{player}):* The game should have only one player. If any level have a different value, the score will be zero.
  - *Number of Winning Objects ($N_{winning}$):* The number of the winning objects should be equal, unless one of the winning objects have "Create" behavior.

- *Box Line Score (B_{score}):* It is similar to Taylor and Parberry metric(**?**) to find the farthest state. This metric cal-

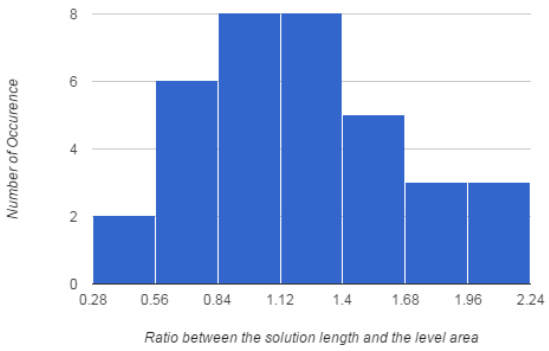

Figure 4: Histogram for the ratio between the solution length and the level area

culates the number of unrepeated moves found in the solution and divide it by the solution length. The following equation represents it:

$$B_{score} = \frac{L_{unique}}{L}$$

where L_{unique} is the number of unrepeated moves in the solution and L is the solution length.

- *Applied Rule Score (R_{score}):* The ratio between the number of applied rules to the solution length should be used for indicting good level design. To find the best ratio between both, We analyzed 40 hand crafted levels from 5 different games and a histogram is plotted in Figure 5. The histogram seems to follow a Normal Distribution with $\mu = 0.417$ and $\sigma = 0.128$. Based on that the Applied Rule Score can be expressed by the following equation:

$$R_{score} = Normal(\frac{R_{app} \pm R_{none}}{L}, 0.417, 0.128)$$

where $Normal(ration, \mu, \sigma)$ is a normal distribution function, R_{app} is the number of applied rules, R_{none} is the number of applied rules without any previous actions, and L is the solution length.

- *Exploration Score (E_{score}):* The increase in the number of explored states by the automated player means that the current level is not obvious to be solved directly by the automated player heuristics. The following equation express this idea:

$$E_{score} = \begin{cases} 0.75 + \frac{N_{exp}}{N_{max}} & \text{solution exists} \\ 0.5 & \text{no solution, } N_{exp} = N_{max} \\ 0 & \text{no solution, } N_{exp} < N_{max} \end{cases}$$

where N_{exp} is the number of explored states and N_{max} is the maximum number of states the automated player can explore.

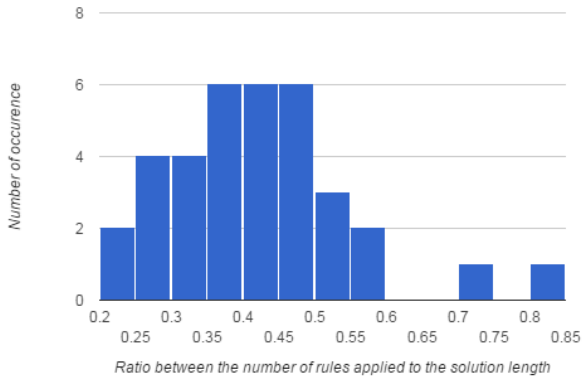

Figure 5: Histogram for the number of rules applied to the solution length

Results and Evaluation

The generated levels are published on a website [1] to collect human feedbacks. The following subsection analyzes the results of the new automated player and compares its results with the original one, followed by the results of the level generation techniques and comparing it with human feedbacks.

Tested Games

Our system is tested against five different games. The five games are completely different to cover different object behaviors and winning conditions. These games are:

- *Sokoban:* The goal of the game is to place every single crate over a certain position. Player can push crates to achieve that goal.
- *LavaGame:* The goal of the game is to reach the exit. The path towards the exit is always blocked by a lava. Player should push crates over the lava to clear his way.
- *BlockFaker:* The goal of the game is to reach the exit. The path towards the exit is always blocked by lots of crates. Player should push these crates to align them vertically or horizontally. Every three aligned crates are destroyed which clear the path towards the exit.
- *GemGame:* The goal of the game is to place at least one gem over one of several locations. Player can create gems by pushing crates. Every three aligned crates are replaced with a single gem instead of the middle crate.
- *DestroyGame:* The goal of the game is to clear every single gem. Gems can be destroyed when they are aligned with two other crates vertically or horizontally. Player should push crates to reach that goal.

Automated Player

Forty handcrafted levels from the five different games were used to compare the new player with original one. Levels are designed with different sizes and ideas to cover different design aspects. Both players plays all the forty levels and reports the solution length and the number of

[1] http://www.amidos-games.com/puzzlescript-pcg/

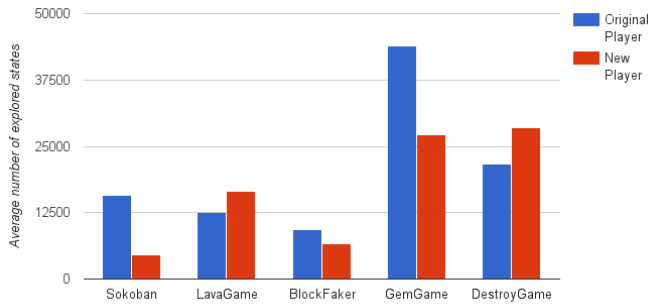

Figure 6: Comparison between the number of explored states for different automated players

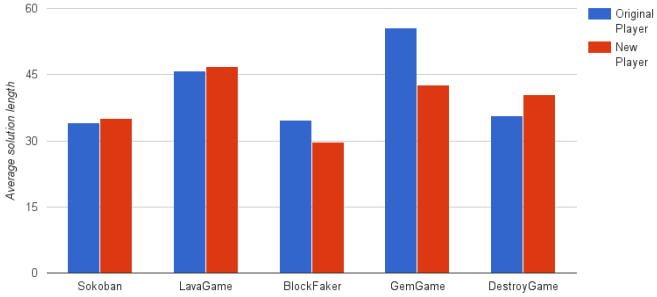

Figure 7: Comparison between the average solution length for different automated players

states explored. Figure 6 shows the average number of states each player explores in each game to reach the goal. The new player outperforms the original player in Sokoban and GemGame, but its almost the similar in the rest of the games. The new players performed badly in DestroyGame and LavaGame. The main reason behind the bad performance in these games is the presence of the Destroy behavior as the core mechanic of the game.

Figure 7 shows the average solution length for each game for the different players. The new player produces a slightly shorter solutions than the original player. Comparing both Figure 6 and Figure 7 a correlation can be noticed between both of them except for Sokoban. Sokoban does not follow the same pattern due to being abstract as the game has a very small amount of objects and just one rule. This abstraction is the main reason both players can reach the goal in almost the same amount of steps.

Level Generation

This section shows the results of the level generation techniques. The new automated player is used with a limit of 5000 explored states to ensure fast execution. Figure 8 shows the eight different level layouts are used to generate levels which are the same like handcrafted games.

All the scores are from all the following techniques are combined and a general correlation is calculated. Figure 9

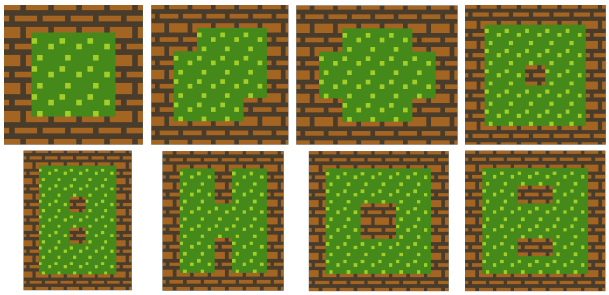

Figure 8: Different level layouts for level generation

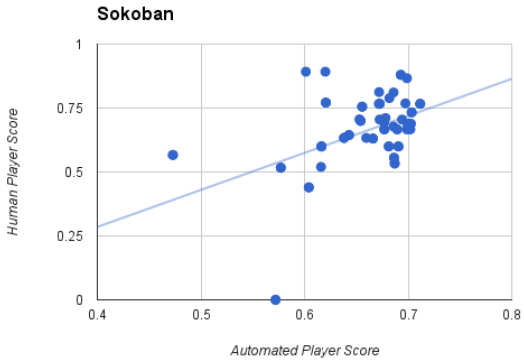

Figure 9: Correlation between all automated player score and human scores for all games

shows the correlation between the system score and human score for all games and techniques.

Constructive Approach One hundred level are generated using the constructive algorithm described in Section **??**. Each level is evaluated and the best two levels are selected. Out of the 80 levels only 15% are reported as unplayable by the system. By testing these levels by human players only 10% are completely unplayable while the remaining levels are very difficult levels (needs more states to be explored).

Genetic Approach The genetic approach gives the system the ability to modify the generated levels to improve their playability and challenge. GA is used for 20 generations with a population equal to 50 chromosomes. The crossover rate is around 70% and the mutation rate is around 10%. GA is applied on each level layout and the best two chromosomes from each layout are selected. Elitism is used with probability equals to 2% to ensure best chromosome doesn't disappear with generations.

- *Random Initialization:* Out of the 80 levels only 75% of the generated levels are playable although the automated player reported only 73.75%.
- *Constructive Initialization:* Using constructive algorithm to initialize the GA increase the overall level playability from 90% to reach 100% by expanding the search space to find better levels than the constructive approach.

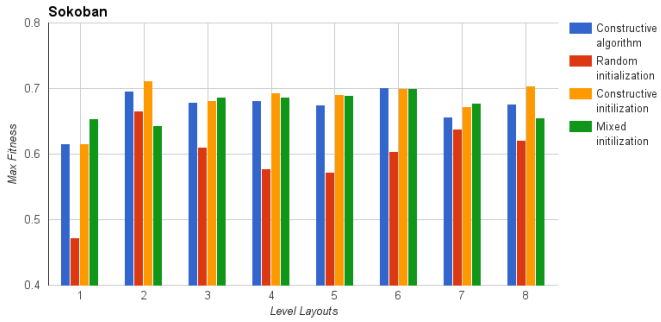

Figure 10: Max fitness of all proposed techniques for Sokoban

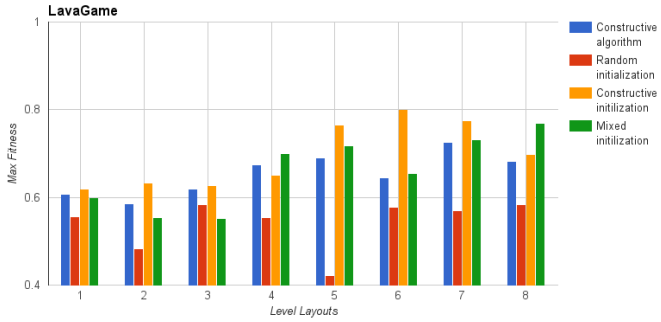

Figure 11: Max fitness of all proposed techniques for LavaGame

- *Mixed Initialization:* Same like constructive initialization with 100% playability and more diverse levels.

Fitness Comparison The following figures shows a comparison between the max fitness of the all presented techniques for Sokoban(Figure 10), LavaGame(Figure 11), and BlockFaker(Figure 12). GA with constructive initialization have the highest score in almost all games, followed by GA with mixed initialization which is almost the same as GA with constructive initialization. The worst of them all is the GA with random initialization as it need more generations to find a good playable levels. Sokoban scores in Figure 10 are almost the similar with all techniques due to simplicity of the game rules and the small number of objects needed to have a playable level.

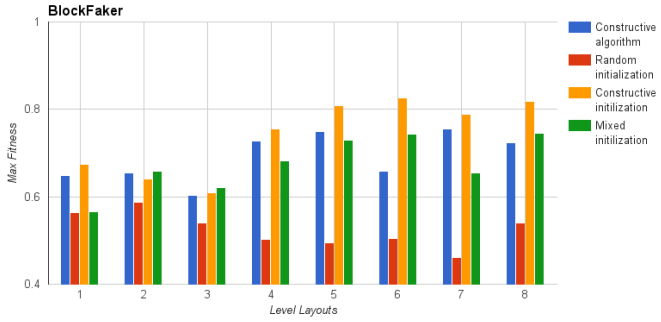

Figure 12: Max fitness of all proposed techniques for BlockFaker

Conclusion and Future Work

This research presented a system to generate levels and rules for Puzzle Script. Also, it proposed several metrics to evaluate puzzle levels and games based on their solution sequence.

The proposed system generates levels regardless of the game rules. It uses two different techniques (Constructive and Genetic approach). The constructive approach resulted in 90% playable levels which is enhanced in the genetic approach to reach 100%, but it needs more time. Genetic approach uses GA with three different initialization methods (Random initialization, Constructive initialization, and Mixed initialization). Random initialization produces levels with different configuration from the constructive approach, but with low playability equals to 75%. The constructive approach produces levels with playability reaching 100%, but with similar structure to the constructive approach. The mixed initialization is similar to constructive initialization in terms of playability but it expands the search space.

The generated levels are tested using human players and a score is given for each level. Comparing the human scores with the system scores indicate a high correlation. This high correlation is a good indication that the proposed metrics can actually measure game playability and challenge. The correlation is higher in some games such as BlockFaker and Sokoban due to the high performance of the automated player in playing them.

This work is a first stone in general level and rule generation. There is a plenty to be done to expand and enhance it. As for future work, we aim to:

- analyze the effect of each metric on the level generation.
- utilize the metrics to analyze the search space for level generation.
- test different techniques rather than plan GA to increase the level diversity like in Sorenson and Pasquier work(**?**).
- improve the time and the quality of the automated player to decrease the generation time.
- generate levels with a specific difficulty.

Acknowledgments

I would like to express my deepest gratitude to Micheal Cook. His wok on ANGELINA was my main inspiration to start working on Procedural Content Generation. I would like to thank my supervisor Prof. Magda Fayek for all the support, guidance, and extreme patience she provides. Without her support this work would not have seen the light. Also Thanks to all my friends for the support and the huge help in collecting human feedback on the results.