Deceptive Level Generator

Adeel Zafar, Hasan Mujtaba, Mirza Omer Beg, Sajid Ali
Riphah Int. University and NUCES-FAST, Islamabad
Email: adeel.zafar@riphah.edu.pk, hasan.mujtaba@nu.edu.pk, omer.beg@nu.edu.pk, sajid.ali123@yahoo.com

Abstract

Deceptive games are games where rewards are designed to lead agents away from a global optimum policy. In this paper, we have developed a deceptive generator that generated three different type of traps including greedy, smoothness and generality trap. The generator was successful in generating levels for a large set of games in the General Video Game Level Generation Framework. Our experimental results show that all tested agents were vulnerable to several kinds of deceptions.

Introduction

Creating manual content for games is a time consuming (Togelius et al. 2010) and expensive task. Delegating content generation to an algorithmic process can save time and money. Procedural Content Generation (PCG) (Shaker et al. 2010) is such a method, where the algorithmic process is used to create a large variety of content including levels, maps, textures, and weapons. The recent advancement in the field of PCG has seen the rise of two different types of algorithms for the purpose of automatic generation: Constructive techniques and Search-Based techniques. With the constructive technique, content is generated in a single pass without any further iterations. Constructive techniques (Shaker et al. 2010) are a simple and fast means of content generation. By contrast, Search-Based techniques (Togelius et al. 2010) regenerate the content in order to improve their quality. Those techniques mostly use an evolutionary algorithm or similar method for content generation.

In a recent study, author (Anderson et al. 2018) introduced the concept of deceptive games. These games are designed in a way to trap the agents or controllers playing the game. The motivation for designing these type of games is focused on a more broader aspect of designing difficulty or challenge in games for Artificial Intelligent (AI) agents. The study focused on providing a Video Game Description Language (VGDL) (Schaul 2013) in the General Video Game AI (GVG-AI) (Perez et al. 2016) framework. In this paper, we have built on this idea and have created a deceptive generator that generates the deceptive levels for a large set of games in the GVG-AI framework. The deceptive generator generates three different types of traps including greedy trap, smoothness trap and generality trap. The experimentation results highlighted the fact that the generated levels of the deceptive generator were challenging for different AI agents.

The remainder of the paper is organized as follows. In section 2, we give some background and related work. Section 3 explains the strategy and algorithm for level generation. Section 4 presents the experimental evaluation. Finally, we conclude the paper in section 5.

Background

Procedural Content Generation

PCG is a technique of generating gaming content, including levels (Dahlskog et al. 2014) (Adrian et al. 2013), maps (Togelius et al. 2010), music (Jordan et al. 2012), racing tracks (Kemmerling et al. 2010), weapons (Hastings et al. 2009), and terrains (Frade et al. 2012) automatically through some pseudo-random process. Though PCG is not a silver bullet for game designers, it has been widely used for Rogue-like games by Indie game developers. Automated content generation techniques can help Indie game developers to limit the cost and time of development which is showcased by successful games such as No Mans Sky (https://www.nomanssky.com/), Binding of Isaac (http://store.steampowered.com) and Faster than Light (http://store.steampowered.com).

VGDL, GVG-AI and GVG-LG Framework

Video Game Description Language (VGDL) (Schaul 2013) was designed by Stanford General Video Game Playing (GVGP). VGDL is a simple, description language to define a variety of 2D games. General Video Game AI (GVG-AI) (Perez et al. 2016) framework is the basis for general video game playing competition, where participants can create different agents and can test them against a variety of games. General Video Game Level Generation (GVG-LG) (Khalifa et al. 2016) track is built on top of GVG-AI. The framework allows participants to create generators that can generate levels for a set of different games. Initially, the framework is composed of three sample level generators including random, constructive and search based generators.

Deceptive Games

In a recent study (Anderson et al. 2018), the author introduced the concept of deceptive games. These games were designed in accordance to lead the agent away from a global optimum policy. To showcase the vulnerability of game playing agents, a number of deceptions were designed. Each trap was focused on a certain cognitive bias. The results showed that different game playing agents had different weaknesses. In this paper, we have built on the existing work and have created a deceptive generator. The deceptive generator generates multiple cognitive traps including greedy trap, smoothness trap and generality trap. For the aforementioned problem, the GVG-LG framework was used and the algorithm was successful to generate a variety of traps for a large set of games.

Method

Classification of Games

In order to implement deceptions in the game set of GVG-LG framework, we have thoroughly analyzed all the 91 games. The initial step of exploring all games exhaustively was important as we had to limit the games, where our implemented deceptions would not make some sense. While exploring the games of the GVG-LG framework, we identified that there are some games perfectly correspond to our deceptive algorithm including zelda, whakmole, and roguelike. There were overall 31 games that are relevant to our deceptions and hence our algorithms give the best result on that. All the rest of the games do not respond positively to our deceptive algorithm. These games had problems such as unplayability of levels, resource deficiency issues and lack of requisite environment.

Deceptive Generator

The deceptive generator generates three different types of traps including smoothness trap, generality trap, and greedy trap. The overall generation work in four different steps including initialization, greedy trap, smoothness trap and generality trap. Before explaining the details of the algorithm, we would explain the concepts that are necessary to understand the working of each algorithm. These concepts are as follows:

- **Game Description**: This file specifies how the game is played and all the interactions.
- **Sprites**: Sprites are the main game elements. They are defined in the SpriteSet section of the game description file (VGDL). In the SpriteSet section, sprites have a name and a type and some parameter values.
- **HashMap**: The HashMap function returns a hashmap that helps decode the current generated level.

Algorithm 1 extracts all the sprite data by the game object. Further to this different ArrayLists have been filled by the appropriate sprite type. In algorithm 1, (from line 1 to 3) are responsible for extraction of sprite data, hash map and key sets associated with each sprite type. The next phase of the algorithm (from line 4 to 12) assigns each sprite with its type and populates the ArrayList.

Algorithm 1: Fill Up ArrayList

```
1  SpriteData = currentGame.getAllSpriteData();
2  HashMap = currentGame.getLevelMapping();
3  KeySet = HashMap.getKeySet();
4  foreach key in KeySet do
5      TempArrayList = HashMap.get(key);
6      SpriteName = TempArrayList.get(spriteIndex);
7      foreach sprite in SpriteData do
8          if sprite.name == SpriteName and
             sprite.TypeMatch then
9              ArrayList.Put(key);
10         end
11     end
12 end
```

Algorithm 2: Generate Greedy Trap

```
   /* call Initialization Routine..
      This routine would be called in
      other traps.                    */
1  GRIDSIZE = 12 ;
2  gridChoice = generateRandom(1,2);
3  avatarPos = generateRandom(1,2);
   /* Initialization Routine ends..  */
4  if gridChoice == 1 then
5      stripRow = GRIDSIZE/4 ;
6      foreach rows >= 1 and rows <= GRIDSIZE do
7          foreach cols >= 1 and cols <= GRIDSIZE
             do
8              if rows = stripRow AND avatarPos = 1
                 then
9                  while cols=keysLength and keyType
                     =avatar OR goal OR wall do
10                     grid + = keyType.get()
11                 end
12             end
13             else if rows < stripRow AND rows > 1
                 then
14                 while keys.Type = resourceKeys
                     AND harmfulKeys do
15                     grid + = keyType.get()
16                 end
17             end
18             else
19                 while keys.Type=resourceKeys do
20                     grid + = keyType.get()
21                 end
22             end
23         end
24     end
25 end
```

The first trap implemented in our generator is the greedy trap. Greedy trap aims to maximize some immediate reward and rely on the assumption that a local optimum would guide

them to the global optimum. We took this notion into consideration and designed a greedy trap. Algorithm 2 incorporates the greedy trap. Initially, an initialization routine (from line 1 to 3. Note that this routine would be called in other algorithms as well) has been called to define the size of the level. The algorithm then divides the level into two parts: one slightly larger than the other. After dividing the level into two parts, we place the items/sprites excluding harmful sprites in the larger section of the level (from line 4 to 9). In the narrow section of the grid, our algorithm places harmful sprites along with collectible sprites to make the greedy trap more effective.

Algorithm 3: Generate Smoothness Trap

```
   /* call Initialization Routine    */
1  if gridChoice == 1 then
2      medianRow = GRIDSIZE/2 ;
3      foreach rows >= 1 and rows <= GRIDSIZE do
4          foreach cols >= 1 and cols <= GRIDSIZE
             do
5              if rows = medianRow and avatarPos=1
                 then
6                  while cols=keysLength and keyType
                     = avatar OR goal OR wall do
7                      grid + = keyType.get()
8                  end
9                  else if rows < medianRow then
10                     generateRandomStrips;
11                     while keys.type=collectibles do
12                         grid+ =keyType.get();
13                     end
14                 end
15                 else if rows > medianRow then
16                 end
17                 generateRandomStrips;
18                 while keys.type=collectibles AND
                     harmful do
19                     grid+ =keyType.get();
20                 end
21             end
22         end
23     end
24 end
```

Smoothness trap exploits the assumption that AI techniques rely on that good solutions are close to other good solutions. This assumption could be exploited by using a mechanism that hides the optimal solution close to bad solutions. In algorithm 3, we have implemented the idea of smoothness trap. Contrary to the greedy trap, in smoothness algorithm, we divided the level into two segments. One segment of the level was implemented as a smooth path and the other as a harsh path. The smooth path (line 6-9) has a low level of risk and hence avatar is positioned close to collectible and goal sprites. On the other hand, harsh or difficult path (line 15 to 20) is implemented as a long path with both collectibles and harmful sprites.

Algorithm 4: Generate Generality Trap

```
   /* call Initialization Routine    */
1  firstPart = GRIDSIZE/3;
2  secondPart = ((GRIDSIZE)-firstPart)/2;
3  thirdPart = (GRIDSIZE)(firstPart+secondPart);
4  gameLevel = getGameLevel();
5  foreach rows >= 1 and rows <= GRIDSIZE do
6      foreach cols >= 1 and cols <= GRIDSIZE do
7          if gameLevel=1 OR gameLevel=2 then
8              if rows=firstPart then
9                  avatarKeys.get();
10             end
11             else if rows=secondPart then
12                 collectibleKeys.get();
13             end
14             else if rows=thirdPart then
15                 harmfulSprites.get();
16             end
17         end
18         else if gameLevel= 3 then
19             //Place harmful sprite with goal
20         end
21     end
22 end
```

Generality trap exploits the concept of surprise by providing a game environment, where a rule is sensible for a limited amount of time. In algorithm 4, we have generated the generality trap. It is worthwhile mentioning that in order to execute a generality trap, the agent or controller has to play at least three levels. The first two levels develop the concept of the agent while the third showcases the surprise element. The algorithm first calls the initialization function and then divides the level into three parts (from line 1 to 3). After the initialization step, the algorithm works for each level separately. If the level is 1 or 2, the algorithm would keep harmful sprites away from goal sprites so that avatar has no experience of combat and it should develop the concept that fighting with harmful sprites is a useless activity. However, when the 3rd or final level was being played, the algorithm places the harmful objects in the vicinity of goal object to surprisingly break the previously developed concept.

Experimentation

In order to showcase our results, we have generated levels for multiple games. The description of these games are as follows:

- **Zelda**: The avatar has to find a key in a maze to open a door and exit. The player is also equipped with a sword to kill enemies existing in the maze. The player wins if it exits the maze, and loses if it is hit by an enemy. Refer to figure 1 for generated levels of zelda.
- **CatPult**: The main theme of this game is to reach the exit door to win. The avatar cannot step on ponds of water,

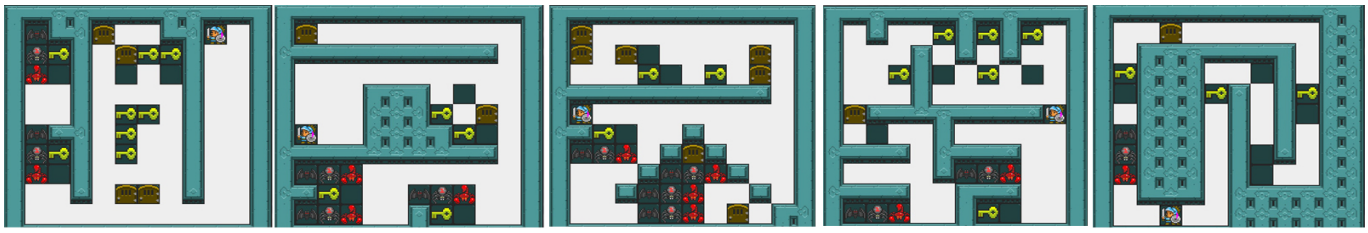

Figure 1: The figure depicts Zelda game levels. From left to right: generality trap level 1, generality trap level 2, generality trap level 3, smoothness trap and greedy trap (Note lower boundary is cropped).

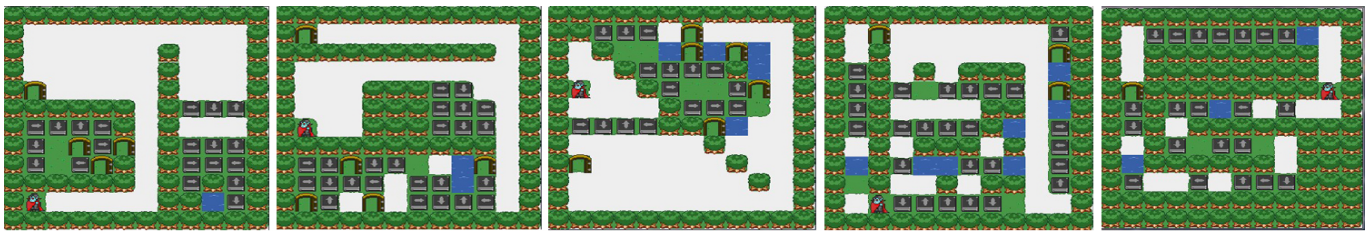

Figure 2: The figure depicts Catapult game levels. From left to right: generality trap level 1, generality trap level 2, generality trap level 3, smoothness trap and greedy trap.

Figure 3: The figure depicts Cakybaky game levels. From left to right: generality trap level 1, generality trap level 2, generality trap level 3, smoothness trap and greedy trap (Note lower boundary is cropped).

Figure 4: The figure depicts DigDug game levels. From left to right: generality trap level 1, generality trap level 2, generality trap level 3, smoothness trap and greedy trap.

however can jump over them using catapults. Each catapult can be used only once. Refer to figure 2 for generated levels of CatPult.

- **Cakybaky**: In cakybaky, you have to bake a cake. To do this task, there are several ingredients that must be collected in order and to follow the recipe. There are angry chefs around the level that chase the player, although they only care about their favorite ingredient, so only the ones that prefer the next ingredient to be picked up are active at each time. Refer to figure 3 for generated levels of Caky-Baky.
- **DigDug**: The objective of this game is to collect all gems and gold coins in the cave, digging its way through it. There are also enemies in the level that kill the player on collision. Refer to figure 4 for generated levels of DigDug.
- **SolarFox**: The main theme of this game is to collect all the diamonds on the screen and avoid the attacks of enemies. The brake of the spaceship controlled by the player is broken, so the avatar is in constant movement. Refer to figure 5 for generated levels of SolarFox.

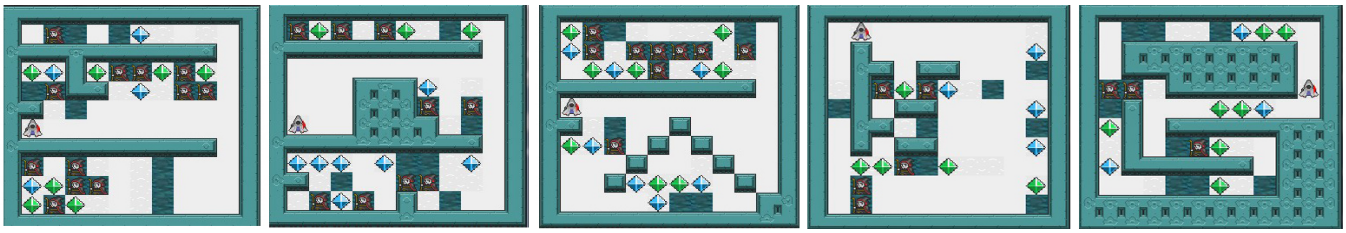

Figure 5: The figure depicts SolarFox game levels. From left to right: generality trap level 1, generality trap level 2, generality trap level 3, smoothness trap and greedy trap.

Table 1: Performance of controllers for generated levels.

Controllers Performance					
Agents	**Algorithm**	**Win Percentage**	**Normalized Mean Score**	**Weakness**	**Best Performing Game**
Number27	Portfolio	52%	0.60	Smoothness Trap	DigDug
thorbjrn	MCTS	40%	0.71	Greedy Trap	Zelda
OLETS	Optimistic Planning	40%	0.63	Smoothness/Greedy Trap	CataPults
NovelTS	Tree	28%	1.0	Greedy Trap	Zelda
sampleRS	Random	28%	0.96	Greedy Trap	CataPults
sampleRHEA	EA	28%	0.80	Greedy Trap	CakyBaky
NovTea	Tree	20%	0.40	Generality/Greedy Trap	CataPults
CatLinux	GA	20%	0.33	Generality/Greedy Trap	CataPults
sampleMCTS	MCTS	16%	0.45	Greedy Trap	CakyBaky
sampleOneStep	One State Lookup	8%	0.13	All	Zelda

The generated levels were tested on a wide variety of agents including OLETS, sampleMCTS, sampleRHEA, sampleRS, NovTea, NovelTS, Number27, sampleOneStep, CatLinux and thorbjrn. Most of the agents were collected from the GVG-AI competitions and some are advanced sample controllers. The agent selection was based on the unique features of their algorithms. Each agent was run multiple times on each level generated by our deceptive generator. The results of the experimentation are shown in table 1. Note that the agents are ranked according to their win rate and mean score. In total, 250 levels were played by 10 different controllers. No single algorithm was able to solve all the deceptions. The Number27 agent was the most successful in accordance with win percentage and the onesteplookahead agent was the least successful of all. It is important to note here that the majority of the agents performed well on zelda. However, no agent was able to successfully play levels of SolarFox. In addition, we can see from table 1 that all the game playing controllers had a different weakness (generality, smoothness or greedy trap).

Conclusion and Future Work

Deceptive games are designed to move agents away from a global optimum policy. In this paper, we have created a deceptive generator that generates different types of traps including generality, smoothness and greedy trap. The generator was successful in generating a large variety of levels for a set of games. In order to test our generator, we generated example levels for five different games including zelda, catapults, cakybaky, solarfox, and digdug. In addition, ten different controllers were tested. The results indicated that each type of deception had a different effect on the performance of agents. No single agent was able to solve all type of traps. The best among all was Number27 and least of all was onesteplookahead.

In the future, we plan to create more traps within our deceptive generator. Preferable, for games where the included three traps are not suited. Another important future step is the creation of agents or controllers which can solve maximum traps posted by our deceptive generator.

Acknowledgment

We acknowledge Riphah International University for support of this research.

References

Anderson, D., Stephenson, M., Togelius, J., Salge, C., Levine, J., Renz, J. (2018, April). Deceptive games. In International Conference on the Applications of Evolutionary Computation (pp. 376-391). Springer, Cham.

Khalifa, A., Perez-Liebana, D., Lucas, S. M., Togelius, J. (2016, July). General video game level generation. In Proceedings of the Genetic and Evolutionary Computation Conference 2016 (pp. 253-259). ACM.

Perez-Liebana, D., Samothrakis, S., Togelius, J., Lucas, S. M., Schaul, T. (2016, February). General video game ai: Competition, challenges and opportunities. In Thirtieth AAAI Conference on Artificial Intelligence (pp. 4335-4337).

Shaker, N., Togelius, J., Nelson, M. J. (2016). Procedural content generation in games. Switzerland: Springer International Publishing.

Dahlskog, S., Togelius, J. (2014, August). A multi-level level generator. In Computational Intelligence and Games (CIG), 2014 IEEE Conference on (pp. 1-8). IEEE.

Adrian, D. F. H., Luisa, S. G. C. A. (2013, August). An approach to level design using procedural content generation and difficulty curves. In Computational intelligence in games (cig), 2013 ieee conference on (pp. 1-8). IEEE.

Schaul, T. (2013, August). A video game description language for model-based or interactive learning. In Computational Intelligence in Games (CIG), 2013 IEEE Conference on (pp. 1-8). IEEE.

Frade, M., de Vega, F. F., Cotta, C. (2012). Automatic evolution of programs for procedural generation of terrains for video games. Soft Computing, 16(11), 1893-1914.

Jordan, A., Scheftelowitsch, D., Lahni, J., Hartwecker, J., Kuchem, M., Walter-Huber, M., ... Preuss, M. (2012, September). Beatthebeat music-based procedural content generation in a mobile game. In Computational Intelligence and Games (CIG), 2012 IEEE Conference on (pp. 320-327). IEEE.

Kemmerling, M., Preuss, M. (2010, August). Automatic adaptation to generated content via car setup optimization in torcs. In Computational Intelligence and Games (CIG), 2010 IEEE Symposium on (pp. 131-138). IEEE.

Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelbck, J., Yannakakis, G. N. (2010, August). Multiobjective exploration of the starcraft map space. In Computational Intelligence and Games (CIG), 2010 IEEE Symposium on (pp. 265-272). IEEE.

Togelius, J., Yannakakis, G. N., Stanley, K. O., Browne, C. (2010, April). Search-based procedural content generation. In European Conference on the Applications of Evolutionary Computation (pp. 141-150). Springer, Berlin, Heidelberg.

Hastings, E. J., Guha, R. K., Stanley, K. O. (2009). Automatic content generation in the galactic arms race video game. IEEE Transactions on Computational Intelligence and AI in Games, 1(4), 245-263.