G-107

Biasing Monte-Carlo Rollouts with Potential Field in General Video Game Playing

Chun Yin Chu†, Tomohiro Harada†, Ruck Thawonmas†

***Abstract* - This paper proposes the use of potential field and biased Monte Carlo rollout in General Video Game Playing (GVGP). Monte-Carlo Tree Search is a famous technique for General Video Game Playing, thanks to its adaptability. However, since the rollouts are performed randomly, it may not be able to search the game state efficiently. Existing research has attempted to bias the rollout by using Euclidean distances to the closest sprites as features, and training the bias weights with Evolutionary Strategy. In this paper, we propose the use of potential field features instead of Euclidean distances as the rollout bias, so as to further improve the performance of Monte-Carlo Tree Search in General Video Game Playing.**

Keywords – General Video Game Playing, Monte-Carlo Tree Search, Game AI, Potential Field

1 . INTRODUCTION

While video game AI has been a popular research field, most game AI programs created by researchers focus on a single game, such as Pac-Man or StarCraft, and can only play one specific game. Such research approach limits the applicability of the proposed methods, as the proposed AI technique is only applied to one single game. To address such problem and to imitate human intelligence, researchers have recently embarked on researches in General Video Game Playing (GVGP) AI, where researchers aim at the creation of AI that can play a wide range of video game, without knowing the game rules beforehand.

Monte-Carlo Tree Search (MCTS) is a famous method in the field of GVGP. Relying on random rollouts to determine the best action, Monte-Carlo Tree Search does not require knowledge of the game rules or any heuristics, thus making itself a widely applicable strategies in various kinds of games. However, due to its stochastic nature, MCTS often cannot search the game space in an efficient manner, and cannot effectively learn the game rules through past experience.

Acknowledging the weakness of MCTS, researchers in the past has attempted to bias the rollouts of MCTS with the use of feature extraction and weight bias. While such attempts have successfully improved the performance of MCTS in GVGP, in this paper, we propose the use of potential field technique in biasing MCTS rollout, which leads to a better performance than the existing method in GVGP.

2. THE GVG-AI FRAMEWORK

The GVG-AI Framework is developed by researchers from University of Essex and other institutions, in order to support the development of game AIs that can excel in many games rather than one specific game [1]. This framework specializes at the creation of classic Atari-2600-style games, including Pac-Man, Space Invaders and Legend of Zelda, putting emphasis on real-time and fast-paced game, rather than turn-based games. Using this framework, the first GVG-AI Competition[1] was organized at the 2014 IEEE CIG conference.

The GVG-AI framework uses the Video Game Description Language (VGDL) to define its games [2]. Featuring a Python-like syntax, VGDL allows game rules to be defined textually, as shown in Fig. 1, which shows how the *Zelda* game is defined in the GVG-AI Framework. In *Zelda*, the avatar has to collect the key on the map and reach the exit, while fighting off monsters with a sword. In the SpriteSet, four categories of sprites are defined for the game, namely *goal*, *key*, *sword*, and *movable* (which

```
BasicGame
  SpriteSet
    goal   > Door color=GREEN img=goal
    key    > Immovable color=ORANGE img=key
    sword > Flicker limit=5 singleton=True img=sword
    movable >
      avatar  > ShootAvatar   stype=sword
        nokey   > img=avatar
        withkey > color=ORANGE img=alien
      enemy >  img=monster
        monsterQuick > RandomNPC cooldown=2
        monsterNormal > RandomNPC cooldown=4
        monsterSlow > RandomNPC cooldown=8

  LevelMapping
    G > goal
    + > key
    A > nokey
    1 > monsterQuick
    2 > monsterNormal
    3 > monsterSlow

  InteractionSet
    movable wall  > stepBack
    nokey goal    > stepBack
    goal withkey  > killSprite scoreChange=1
    enemy sword > killSprite scoreChange=2
    avatar enemy > killSprite scoreChange=-1
    key  avatar   > killSprite scoreChange=1
    nokey key     > transformTo stype=withkey
  TerminationSet
    SpriteCounter stype=goal   win=True
    SpriteCounter stype=avatar win=False
```

Fig. 1. The VGDL rule definition of the *Zelda* game

†Intelligent Computer Entertainment Laboratory, Ritsumeikan University

1. http://gvgai.net/

Fig. 2. A *Zelda* game level defined in textual form, and the actual game generated

includes both *avatar* and *enemy*). The InteractionSet defines how sprites interact with each other; for example, if the sword touches enemy, the enemy will be killed and score will increase, if avatar with no key collides with the key, the avatar will be *withkey* and can approaches the exit. The player wins the game if the avatar *withkey* has "killed" the goal, but loses if the avatar is killed by the monsters, as defined in the TerminationSet.

With the basic game rules defined, the game creator can define the game level using the symbol defined in the LevelMapping in the game rules. If not stated otherwise, "w" represents wall and "A" represents avatar in the level mapping. The game engine, written in Java, will read both the game rules VGDL file and the level file, and generate a playable game level, as illustrated in Fig. 2.

Unlike traditional research in game AI, game rules defined in VGDL are hidden from AI developers in GVGP, who have to develop an AI that can excel in games without knowing the rules beforehand, normally by utilizing the simulation method provided by the framework for learning and decision-making. The nature of GVGP makes non-heuristic method such as MCTS a powerful tool. At the same time, researchers are also exploring techniques which can improve MCTS in a GVGP setting.

3. RELATED RESEARCH

3.1 Fast-Evolutionary MCTS in GVGP

Perez et al. proposed a Fast Evolutionary Adaption for Monte Carlo Tree Search [3], where a weight vector is trained by (1+1) Evolutionary Strategy and applied to bias the rollout of MCTS. In normal MCTS, rollouts are performed randomly, and the reward (or penalty) received at the end of the rollout is back-propagated to update the values of all nodes on the branch. Instead of random rollout, Perez et al. proposed the use of biased rollout, where the probability of which action will be taken in the rollout is determined by a weight vector and feature extracted from the game state. Assuming there are *N* features extracted from a game state *S*, and there are *A* actions available in the current state. The relative strength a_i of each action *i* is calculated as a weighted sum of feature values f_j of each feature *j*:

$$a_i = \sum_{j=1}^{N} w_{ij} f_j \quad (1)$$

After the relative strength of all actions in *A* have been calculated, the softmax function is then used to calculate the probability of taking each action:

$$P(a_i) = \frac{e^{-a_i}}{\sum_{j=1}^{A} e^{-a_j}} \quad (2)$$

The weight vector is trained using (1+1) Evolutionary Strategy, with the reward received at the end of the rollout used as the fitness of the weight. The weight vector w_{ij} is evolved after certain number of rollouts have been performed in MCTS. On the other hand, features being extracted from the game state are hand-coded by the AI developer for the specific game. For example, in Space Invaders, Perez et al. picked the distance between the cannon and the rightmost or leftmost alien as the feature. This proposed method provided good performance in *Space Invaders* and *Mountain Car*.

Perez et al. further improved the Fast Evolutionary MCTS by combining it with knowledge acquisition [4]. The Knowledge-based Fast Evolutionary MCTS (KB Fast-Evo MCTS) is based on the proposed method in [3], but adapted it to the GVG-AI platform. Since the game rules are unknown to the AI developer, the feature extraction can no longer be hand-coded beforehand. Instead, the Euclidean distances to the closest NPC, resource, non-static object and portal are extracted as features for every game state.

The rollout result is not only used to evolve the weight vector, but also to acquire knowledge of the game rules. Through the rollout performed in the past, the AI can know which type of sprite is beneficial (generate a score gain when collided) and which is hostile (generate a score loss when collided). For rollouts that do not result in a score difference at the end, those reduced the distance between the avatar and beneficial or unknown sprites will receive a higher reward. The rollout reward will affect the action selection by MCTS and the evolution of the weight.

Whereas the Knowledge-based Fast Evolutionary MCTS proved itself a vast enhancement ordinary MCTS, there is still room for improvement. While using Euclidean distances as features is simple and intuitive, such features fail to take into account the direction and position of the sprite. A sprite standing left to the avatar and another sprite standing right to it will have the same feature values, provided that the Euclidean distances are the same. The feature extraction employed by Perez et al. oversimplifies the game state, and replacing Euclidean distances with another set of features should provide better result.

3.2 Potential Field in Game AI

The concept of potential field has provided us inspiration for a new feature definition in biased MCTS. Proposed by Khatib, potential field originated as an application in robotics [5]. In the potential field approach, the robot moves in a field of force, where the goal is an attractive pole and the obstacles are repulsive surface. The robot moves through the space by heading towards attractive pole while avoiding repulsions.

Potential field is found useful not only in robotics, but also in game AI. Hagelback applied potential field in StarCraft, a famous computer strategy game, and created an AI that can navigate 2D game maps using a combination of potential field and A* algorithm [6]. In the proposed method, interesting objects are assigned attracting force field and obstacles are assigned repulsing force field. The force field is strongest at its center but generally degrade over distance from the center. After calculating all the effects of potential fields emitted by nearby objects, the agent will then move to the position with the highest potential, meaning the most attracting position. The total potential at the position (x, y) is calculated by:

$$p_{total}(x,y) = \sum_{i=1}^{N} w_i p_i(x,y) \qquad (3)$$

In (3), N is the number of objects that are affecting position (x, y), and w_i is the weight for subfield *i*. Hagelback also proposed an improvement to the formula, which consider only the strongest enemy field, instead of the weighted sum of all enemy field, so as to avoid being stuck in a local optima, and to avoid mistakenly marching into enemy's territory. Nonetheless, Hagelback's work has shown that potential field is useful in navigating 2D game map, and the next section of this paper will propose applying potential field as a bias for MCTS.

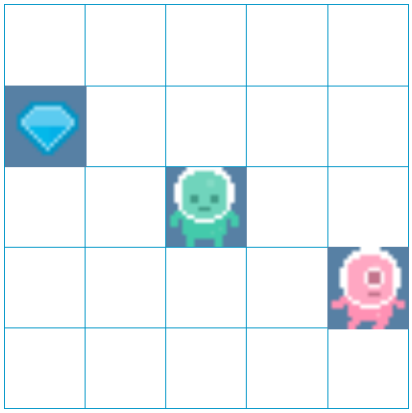

Fig. 3. An example game state in GVG-AI games

4. PROPOSED METHOD

Our proposed method is based on the KB Fast-Evo MCTS proposed in [4], but replaces the Euclidean distance feature with a potential-field-based feature. During the biased rollout, the probability that the avatar will move into a position is determined by its total potential, avatar is more likely to move to a position with higher total potential. The total potential at a position is calculated by aggregating all subfield generated by the closest NPC, resource, non-static object and portal, using a formula similar to (3). Whether a sprite emits attracting force or repulsive force depends on whether it is a beneficial sprite, unknown sprite or a hostile sprite, as determined by the knowledge base. The strength of force of each subfield is trained using Evolutionary Strategy, similar to how Fast Evolutionary MCTS trains its weight vector.

While both methods are based on Perez et al.'s KB Fast-Evo MCTS, our proposed method differs significantly from the Knowledge-based Pathfinding MCTS proposed in [7]. The method proposed in [7] combines Perez et al.'s method with pathfinding algorithm; in the first 50 game steps of each game, the AI will play the game using the exact KB Fast-Evo MCTS as described in [4], and acquire knowledge of the game through the biased rollouts. After that, the AI will switch to pathfinding mode, and identify targets by consulting its knowledge base. On the other hand, our method does not use any pathfinding algorithm. Instead, our method improves KB Fast-Evo MCTS by defining a new set of features and weight vector for the biased rollout. The following paragraphs will detail the feature extraction, weight training and biased rollout in our proposed method.

4.1 Feature Extraction

For each game state, the positions of the closest NPC, resource, non-static object and portal, as well as the position of the avatar, are extracted as features. In most games featured in the GVG-AI framework, the game map

is a 2D grid in which position can be represented as integral coordinates. Thus, the position of every sprite can be easily represented by two integers and stored in memory. In case that the game supports decimal coordinates, the decimal values are rounded down to integers.

For instance, in the game state illustrated by Fig. 3, the avatar is positioned at the center, in between a resource and a pink NPC, which have a type number 1 and 2 respectively. The coordinates of the upper-leftmost corner are defined as (0, 0). The feature extraction function will extract [1: (0, 1)], [2: (4, 3)] and [A: (2, 2)] from the state, meaning that a sprite of type 1 is positioned at (0, 1), sprite of type 2 at (4, 3) and avatar (A) at (2, 2). Later in the biased rollout, potential fields will be assigned to these positions.

A point to be noted here is that, it is easy to calculate the Manhattan distance between two object o_i and o_j using this feature extraction:

$$manDist(o_i, o_j) = |o_i.x - o_i.x| + |o_i.y - o_i.y| \quad (4)$$

In (4), $o_i.x$ and $o_i.y$ represents the x coordinate and y coordinate of object o_i respectively. The Manhattan distance will be used in calculating the force of field at a position in the biased rollout.

4.2 Weight Training

The weight vector *W* contains a decimal weight value w_i for each sprite type *i* in the game. The value of w_i will determine the strength and affecting area of the sprite type *i* during the biased rollout. Thus, the number of dimensions of *W* equals to the number of sprite types in the game.

This weight vector is trained by a (4 + 1) Evolutionary Strategy, which maintains a population of 4 individuals plus 1 mutant. Let *avg(width, height)* donates the average value of the width and height of the game map, the noise value n_j and weight value w_{ij} for sprite type *i* of an individual *j* is initialized by:

$$n_j = \text{avg}(\text{width}, \text{height})/\,5 \quad (5)$$

$$w_{ij} = \text{avg}(\text{width}, \text{height}) + n_j \times gaussianError() \quad (6)$$

The *gaussianError()* function provides a normally distributed decimal value with mean 0.0 and standard deviation 1.0. At the start of the program, 4 individuals will be generated using the above formulae. The fitness is set to negative infinity in the beginning.

Before the first rollout, two parents are selected randomly from the population, in order to generate a new mutant. Each weight value w_{ij} in the new mutant is selected randomly from either of the parents. Mutation is performed by adding a noise value, calculated by n_j X gaussianError(), is to w_{ij} in the new mutant. The initial fitness of new mutant is set to 0. The new mutant is then used in biasing MCTS rollouts, the rewards from which are accumulated in the fitness value of the mutant.

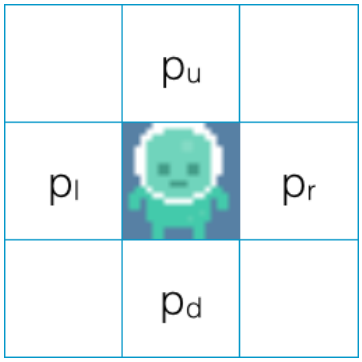

Fig. 4. Neighboring positions of the avatar

After 100 rollouts, an evolution will be performed to update the population and generate a new mutant. If the fitness value, i.e. the accumulated total rewards value of the 100 rollouts, is higher than the fitness of the worst individual in the population, the mutant will replace that individual and the evolution is considered a success; otherwise the mutant will be discarded and the evolution is failed. To generate a new mutant, a random individual and the best individual are selected as parents, following the crossover and mutation procedure described in the previous paragraph. The above process is iterated until the end of the game. The noise value n_j will be increased if the evolution succeeds, and reduced if the evolution fails.

4.3 Biased Rollout

During every step in the biased rollout, the avatar will have to decide which direction (Up, Down, Left or Right) to move to. In random rollout, the direction is selected randomly. However, in our proposed method, the direction is selected according to the total potentials of each adjacent position.

For each sprite type *i* in the feature set *F*, a potential field is created with the position f_i as the center. The strength of the field equals to w_i of the weight vector *W*, and the strength is decreased by 1 as the Manhattan distance from the sprite increases by 1. In other words, given a potential field of sprite *i* centered at p_a and another position p_b, if manDist(p_a, p_b) > w_i, the potential field of sprite *i* will have no effect on the position p_b.

The field type of the sprite is decided by the knowledge base. If past experience has shown that the sprite type generates a score gain, or if there is no record of that type in the knowledge base, it will be assigned an attracting force; on the other hand, if the sprite type begot score loss upon collision in the past, the sprite will be attached a repulsive force. In our implementation, attracting force is represented as positive decimal values (w_i) while repulsive force as negative decimal values ($-w_i$).

When deciding the next direction, the total potential of each neighboring position p_x, as illustrated in Fig 4, are calculated using the following formula:

$$p_{total}(p_x) = \sum_{i=1}^{N} sign(i)|w_i - manDist(p_x, p_i)| \quad (7)$$

In (7), N is the number of objects that are affecting position p_x, and w_i is the weight value for the type *i*. The return value of *sign(i)* is either 1 or -1, and is determined by whether type *i* is hostile, beneficial or unknown in the knowledge base.

After calculating the total potential of all neighboring position, a roulette selection is performed on the four values to determine which position the avatar shall move into. Positions with higher potential have higher chance of being selected. In case there is negative value in the total potential values, $\left|\min_x p_{total}(p_x)\right| + 1$ is added to all potential values to ensure all potential values are positive before the roulette selection.

There are games where the avatar can perform actions other than movements, e.g. in *Zelda*, the avatar can move in four directions and attack approaching enemies; in games like *Seaquest* and *Aliens*, the avatar can shoot at enemies besides moving. In these games, an additional weight for action w_a is added into the weight vector and roulette selection is performed on the value w_a along with all total potential values of neighboring positions. If action is selected, the avatar will perform the action instead of movement in the rollout. The value of w_a is trained by Evolutionary Strategy along with other weight values.

Furthermore, there are games where movement is restricted. For example, in games like *aliens* and *Eggomania*, the avatar can only move horizontally, not vertically. In such cases, the potential values of the corresponding positions (p_u and p_d) are ignored in the roulette selection.

5. EXPERIMENTS

5.1 Comparison with KB Fast-Evo MCTS and MCTS

Experiments are conducted to compare the performance of our proposed method with the Perez et al.'s KB Fast-Evo MCTS and random MCTS. The test procedure is similar to that in [7]. In our experiment, each controller played all games in the training set and validation set of the 2014 GVG-AI Competition, and the controllers' performance in these games were compared. In each set, there are 10 games, each of which has 5 levels. For every game level, each controller will play the game until the game ends, meaning either the controller wins or loses the game, or the time limit (2000 game steps for most games) is reached.

To ensure fairness, each game used the same random seed for all controllers and all levels. The tests were performed in a Mac OS X environment. Similar to the GVG-AI Competition, the controller with the highest win percentage was considered victorious among the three controllers. The average scores were compared instead, if the win percentages were the same.

In our experiment, the source code of KB Fast-Evo MCTS was provided by the main author of [4]. The

TABLE I. EVALUATION RESULTS USING THE 2014 TRAINING SET

	Proposed Method		**KB Fast-Evo MCTS**		**Random MCTS**	
	Avg	***Win_Rate***	***Avg***	***Win_Rate***	***Avg***	***Win_Rate***
aliens	**70.8**	**1**	64.12	1	61.6	0.96
boulderdash	**11.88**	**0.08**	8.16	0	7.32	0.04
butterflies	**25.6**	**1**	24.16	1	28.8	0.96
chases	**3.44**	**0.12**	2.76	0.08	0.84	0
flogs	**-1.12**	**0.24**	-1.28	0.24	-1.36	0.08
missilecommand	3.84	0.56	**4.2**	**0.64**	0.72	0.36
portals	0.2	0.2	**0.24**	**0.24**	0.08	0.08
sokoban	**0.92**	**0.28**	1.04	0.24	0.72	0.08
survivezombies	**34.76**	**0.4**	31.4	0.4	32.12	0.4
zelda	**2.76**	**0.04**	1.68	0	2.28	0.04
Victory	**8**		2		0	

TABLE II. EVALUATION RESULTS USING THE 2014 VALIDATION SET

	Proposed Method		**KB Fast-Evo MCTS**		**Random MCTS**	
	Avg	***Win_Rate***	***Avg***	***Win_Rate***	***Avg***	***Win_Rate***
camelRace	-0.6	0.16	**-0.2**	**0.4**	-0.68	0.12
digdug	**28.44**	**0**	22.16	0	15.88	0
firestorms	**-0.96**	**0.08**	-1.36	0	-3.08	0
inflection	5.36	0.76	5.48	0.84	**6.36**	**0.84**
firecaster	6.88	0	5.32	0	6.88	0
overload	**14.4**	**0.16**	14	0.08	13.72	0.08
pacman	133.96	0	111.96	0	**152.68**	**0**
seaquest	**289.76**	**1**	530.16	0.88	586.36	0.6
whackmole	**9.52**	**0.84**	3.4	0.44	1.56	0.68
eggomania	**8.56**	**0.16**	2.4	0	1.84	0
Victory	6		1		2	

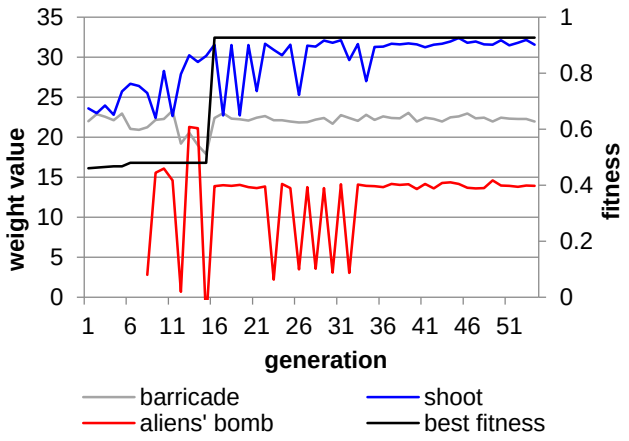

Fig. 5. Changes of weight vector values in an *aliens* game

sampleMCTS package on the GVG-AI Competition official webpage, which was based on Upper Confidence Tree, was used as the random MCTS. The Rollout out depth was set to 10 for all MCTS rollouts.

As illustrated in Table 1 and Table 2, our proposed method achieved a better performance overall. Our method won in 16 games out of 20 games against Perez et al.'s method; also, our method won in 17 games out of 20 games against random MCTS, and drew 1 game. In contrast to random MCTS that performs rollouts randomly, our method used knowledge base and feature extraction to search the game space more effectively. Compared with the Euclidean distance feature used by Perez et al.'s method, the potential-field-based bias used by our proposed method were able to preserve both distances and positions of sprites, without oversimplifying the game state. Therefore, our proposed method is able to outperform both Perez et al.'s method and random MCTS.

5.2 A Closer Look at the Weight Vector in *aliens*

In order to observe how the weight evolves, a closer look at the weight vector in an *aliens* level was observed and plotted in Fig. 5. In *aliens*, illustrated by Fig. 6, avatar gains score by shooting missile at the aliens and the barricades, while avoiding bombs dropped by the aliens. As shown in Fig. 5, the weight for ACTION (shootnig) and aliens' bomb increased through the game, meaning the avatar had higher tendency to shoot missile and avoid aliens' bombs in the rollout, which is a logical tactics in the game. The weight for barricade was maintained at a relatively high level, as the result, the avatar stayed closer to the barricade and shot more often at barricade, leading the higher score gain. Overall, the evolution of weight vector led the avatar to behave in a way that engendered higher score, thus illustrating how evolution of weight vector begets better performance than other method.

6. CONCLUSIONS AND FUTURE WORKS

This paper proposes the use of potential field as bias in Monte Carlo rollouts in a GVGP settings, and experiment

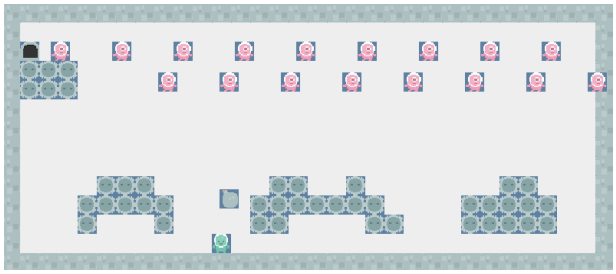

Fig. 6. A Screenshot of an *aliens* game

shows that potential field bias offered an improvement over existing method that used Euclidean distance as feature. In the current stage, our method uses a simple Evolutionary Strategy to train the weight vector. As a future work, we aim at further advancing this method by replacing Evolutionary Strategy with machine learning technique.

ACKNOWLEDGEMENT

This work was supported in part by JSPS KAKENHI Grant Number 15H02939.

REFERENCE

[1] J. Levine, C. B. Congdon, M. Bida, M. Ebner, G. Kendall, S. Lucas, R. Miikkulainen, T. Schaul and T. Thompson, "General Video Game Playing", Artificial and Computational Intelligence in Games: A Follow-up to Dagstuhl Seminar 12191, 2012, pp. 77-83.

[2] T. Schaul, "A Video Game Description Language for Model-based or Interactive Learning", Proceedings of the IEEE Conference on Computational Intelligence in Games, 2013, pp.1-8.

[3] S. M. Lucas, S. Samothrakis and D. Perez. "Fast Evolutionary Adaptation for Monte Carlo Tree Search", Applications of Evolutionary Computation, Springer Berlin Heidelberg, 2014, pp. 349-360.

[4] D. Perez, S. Samothrakis and S. M. Lucas, "Knowledge-based Fast Evolutionary MCTS for General Video Game Playing", Proceedings of the IEEE Conference on Computational Intelligence and Games, 2014, pp. 68-75.

[5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," The International Journal of Robotics Research, 1986.

[6] J. Hagelback, "Potential-field based navigation in StarCraft", Proceednigs of the IEEE Conference on Computational Intelligence and Games, pp.388-393, 11-14 Sept. 2012

[7] Chun Yin Chu, Hisaaki Hashizume, Zikun Guo, Tomohiro Harada, Ruck Thawonmas, "Combining Pathfinding Algorithm with Knowledge-based Monte-Carlo Tree Search in General Video Game Playing," Proc. of the 2015 IEEE Conference on Computational Intelligence and Games (CIG 2015), Tainan, Taiwan, Aug. 30 - Sep. 2, 2015.