Scripting the Game of *Lemmings* with a Genetic Algorithm

Graham Kendall
School of Computer Science & IT
University of Nottingham
Nottingham NG8 1BB, UK
Email: gxk@cs.nott.ac.uk

Kristian Spoerer
School of Computer Science & IT
University of Nottingham
Nottingham NG8 1BB, UK
Email: kts@cs.nott.ac.uk

***Abstract*— The game of *Lemmings* has been offered as a new Drosophila for AI research. Yet the authors have no knowledge of any efforts to create a computer player for *Lemmings*. This paper presents a scaled down version of *Lemmings* and offers an evolutionary approach to solving maps. Scripts are evolved using a genetic algorithm. Results show that this approach is able to solve increasingly complex maps.**

I. Introduction

John McCarthy [1] states that "*The computer game Lemmings can serve as a new Drosophila for AI research*". The game of chess is traditionally referred to as the Drosophila[1] of artificial intelligence research; a test bed for the development of ideas in computer game playing programs. It has received decades of research effort leading eventually to the creation of Deep Blue [2] that defeated World Champion Garry Kasparov in 1997. Yet, despite McCarthy's introduction of *Lemmings* as a new problem, the authors have no knowledge of any efforts to create a computer player for this game.

This paper presents a scaled down version of *Lemmings* and offers an evolutionary approach to solving maps. Section II introduces the original *Lemmings* game, section III provides a background of prior efforts to make computers play games, section IV describes the experimental method and section V discusses the results. These results demonstrate that an evolutionary approach is a suitable method for developing a computer program to play *Lemmings*. Finally, section VI offers suggestions for future research.

The work presented is a first attempt at creating a computer player for the game of *Lemmings*.

II. The Game of Lemmings

The computer game *Lemmings*[2] is a 1-player game displayed on a screen as a 2-dimensional image. It consists of entities called lemmings that move around an environment.

The environment contains an entrance that admits the lemmings and an exit that offers their only safe means of escape. Static platforms and walls are arranged to make navigation from entry to exit difficult for the lemmings. Hazards exist in the form of fire, water and rotating blades, which kill lemmings that touch them.

Lemmings are walkers by default. Walker lemmings on a platform move by default to the right, they cannot stand still, and will swap direction if their path is blocked. If a walker lemming approaches the end of a platform, it will continue over the edge and fall. A walker lemming that falls too far will die when it lands on another platform.

There are eight special types that can be assigned to the lemmings. For each map there are a set number of each type available.

1) A **basher** makes a hole in a vertical wall and turns back into a walker once it has passed through.
2) A **blocker** stands still and blocks the path for other lemmings, making them change direction. It remains a blocker until the end of the game, and is assumed to be sacrificed.
3) A **bomber** counts down five seconds then explodes and dies, destroying the environment immediately surrounding it. It is assumed to be sacrificed.
4) A **builder** lemming builds a diagonally upward bridge for a set count of units or until a solid object blocks the way, at which point it turns back into a walker.
5) A **climber** moves in the same manner as a walker lemming but climbs up vertical walls. It remains a climber until the end of the game (climbing any wall it comes into contact with) and can also be made into any of the other seven special lemming types.
6) A **digger** makes a hole in a floor (similar to a basher) and turns back into a walker once it has passed through.
7) A **floater** moves in the same manner as a walker lemming but can survive drops of any height. It remains a floater until the end of the game (surviving all drops) and can also be made into any of the other seven special lemming types[3].
8) A **miner** makes a diagonally downward hole (similar to a basher and a digger) and turns back into a walker once it has passed through.

Lemmings that are still active in a map can be nuked to blow them all up. Nuking is necessary to clear lemmings from the

[1] A fruit-fly much used as an experimental subject in the study of genetics, *Oxford English Dictionary*.

[2] *Lemmings*™ is a trademark of Psygnosis Limited.

[3] A lemming that is both a **climber** and **floater** is called an *athlete*.

map, for example lemmings that are blocking, since a game will not end until the map is empty of all lemmings.

The lemmings possess a number of subtle behaviours that a human player learns through experience. For example, a miner can be used to mine the ground under a blocker lemming causing the blocker to fall and change back into a walker, multiple builder lemmings can be used to build a zig-zag staircase up a hole, and a lemming can be blocked before setting it to a bomber in order to increase accuracy. These subtle behaviours, that make the game more challenging, are not discussed in this paper.

Further complexity is introduced to the game with walls and platforms that cannot be destroyed (by basher, digger, miner or bomber lemmings) and walls that can only be bashed through in one direction.

The objective of the game is for a specified number of the lemmings that enter the map to safely navigate to the exit before a predefined time limit. If the player forms a strategy to successfully complete a map then they advance to the next map, which requires a more complex strategy to save the required number of lemmings.

A. Problems posed by Lemmings

John McCarthy [1] highlights various phenomena of the game of *Lemmings* that liken it to real world problems. It runs too fast and contains too much information to be fully formalised by the player, so only part of the information is extracted from the map. Moreover, maps are typically larger than the screen, so hidden areas exist off-screen and knowledge of the game state is partial. The player's appreciation of the number of pixels a lemming moves per second is no more than an approximation, and groups of lemmings are not individualised, but formalised by the player as groups.

The game of *Lemmings* also raises game-playing issues. It is deterministic: the same actions in identical situations will always lead to the same outcome. A strict time limit is set for each map so navigation needs to be quick. The authors have observed human players and noted that strategies were formed before the game was started. This suggests that a *planned* approach is used rather than a *reactive* one, most likely a result of the static nature of the environment. If there were dynamics to the environment, for example a randomly moving exit, then this might not necessarily be the case.

The game updates in *real time* so strategies need to be timed correctly. In fact, an action made a pixel out of place might ultimately result in failure. A program to play *Lemmings* must therefore be fast enough to make actions in real time. Also, it is often better for the player to do nothing and allow events to unfold automatically whilst awaiting the opportunity to employ a strategy.

Game playing programs are a fruitful area for artificial intelligence research and the issues described make *Lemmings* an interesting game to use. Yet the authors have no knowledge of a computer program to play *Lemmings*. The following section describes important research into programs to play other games.

III. Computers playing Games

Considerable effort has been spent developing computer programs that play games of skill. Chess has received the most interest and there has been a sustained effort since Turing [3] and Shannon [4] pioneered work in the 1950's. Work on Chess continued through to the 1980's [5], [6], [7] and the 1990's when Hsu et al [8] gained success with Deep Thought. Eventually Deep Blue [2] defeated World Champion Garry Kasparov in 1997. Despite Deep Blue's success, the program did not improve through learning.

Checkers has also received a large amount of interest from the research community. Samuel [9], [10] layed the groundwork in 1959. Later Schaeffer et al [11], [12] developed an automated Checkers player called Chinook that took the world title from Marion Tinsley who had been champion for over forty years. Chellapilla and Fogel [13], [14] and Fogel [15] co-evolved a checkers program in the late 1990's that was able to defeat expert players. In contrast to Deep Blue and Chinook, that both used large opening and endgame databases, Chellapilla and Fogel's program started with no knowledge of the game and learned strategies through co-evolution. This approach is particularly attractive to programmers because of the time saved by not having to incorporate domain knowledge.

Other games have also received the attention of researchers. Müller [16] details the many difficulties and challenges of computer Go, which has an even larger search space than Chess. Billings et al [17] combined several abstraction techniques to develop an improved poker-playing program that was competitive against a world-class opponent. While Kendall and Willdig [18] have shown that an adaptive poker player is a promising research direction. Backgammon has been tackled by Tesauro [19] who applied Temporal Difference Learning to create a program that was able to beat human players. Moriarty and Miikkulainen [20] evolved Artificial Neural Networks to develop a game playing strategy for Othello, and Sheppard [21] used simulations of likely game scenarios in Scrabble, both of which yielded excellent results. A survey of games solved and receiving interest can be found in Schaeffer [22] and Jaap van den Herik [23].

It is the authors' intention to develop a program to play *Lemmings* to a level where simple, yet non-trivial, maps can be solved.

IV. Experimental Method

A. Game

If experiments are to be conducted on the game of *Lemmings* then a method is required to obtain data from it at each frame. This data shall be used as the input to a program that will output a strategy. This strategy will then be used to play the *Lemmings* game. The authors have developed a graphically simpler version of the game of *Lemmings*. This will provide direct access to both the output displayed by the game and the input required by the game at each frame.

The game includes most of the elements from the original game of *Lemmings* so that the research issues can be

addressed. The graphical simplification means it has a few differences. The original has movement and collision detection at the pixel level, whereas the version used for the experiments has movement and collision detection within a grid. Each grid cell is approximately 10 pixels square. The original game was most likely updated approximately 30 times per second, whereas the version used for the experiments is updated once per second, which results in jerky movement. Gradients do not exist. For this reason the **miner** lemming type is omited from the list of eight special lemming types. The seven remaining types are all implemented.

These differences comprise a scaling-down of map resolution by approximately 10:1 and also of the game update-frequency by approximately 30:1. Also, some complexity is lost because the platforms can all be destroyed, from all directions, and there is no variation to the width of the platforms. Regardless, the scaled-down game is governed by the same rules that were imposed upon the original *Lemmings* game. The game can be scaled-up to match the complexity of the original game of *Lemmings* once a computer player has been developed to play the scaled-down version.

The authors have also developed a map editor, which is used to create, edit and save the test-case maps.

B. Test Set

Figures 1 to 7 show the seven test case maps used in the experiments. 'E' represents the entrance. 'X' represents the exit. Lemmings that drop onto the bottom of the map die. The time-limit for each map is set to 300 seconds. The first map requires the use of only one special lemming type (i.e. a basher). Each successive map requires the use of all those lemming types used so far, plus an additional type. The final map requires the use of all seven special lemming types.

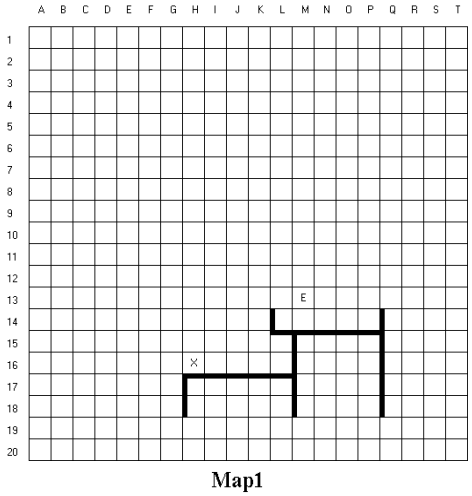

Fig. 1. Showing map1. Bashers are available.

C. Genetic Algorithm

Genetic Algorithms (GAs) [24], [25], [26], [27] are search and optimisation algorithms based on the natural process

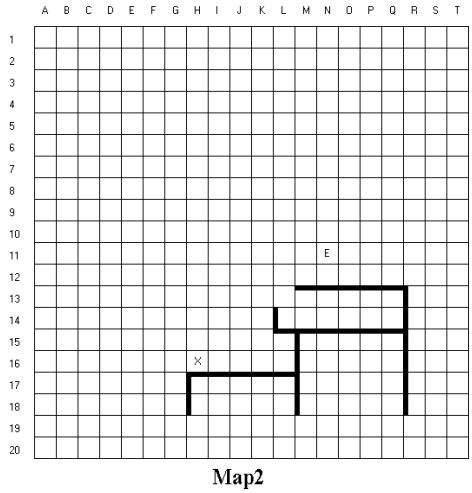

Fig. 2. Showing map2. Bashers and blockers are available.

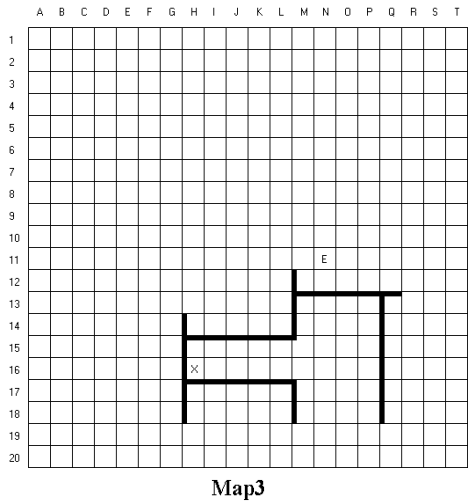

Fig. 3. Showing map3. Bashers, blockers and bombers are available.

of evolution. Alliot and Durand [28] evolved an evaluation function for an Othello program using an enhanced GA and Donelly [29] applied a GA to evolve neural networks used to evaluate Go positions.

The authors use a GA to evolve a strategy that can solve *Lemmings* maps using the following representation.

1) Representation: A strategy can be represented as a script. This script is accessed by a controller. The controller knows only the current time-step t of the game. It has no knowledge of the map or of the lemmings. The controller references the script each time-step and relays an order back to the lemmings. The controller is ignorant of orders that cannot possibly be completed, or are sent to lemmings that do not exist.

The lemmings have no knowledge of the environment around them or any of the other lemmings. They simply listen to the controller. If the controller sends an order to a lemming then it is completed by that lemming alone. A lemming with no order continues on its current path. Lemmings are identified

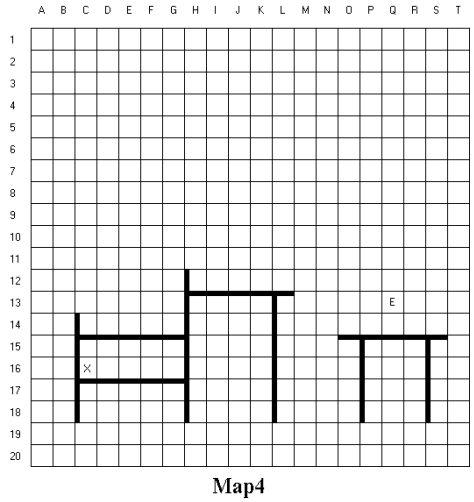

Fig. 4. Showing map4. Bashers, blockers, bombers and builders are available.

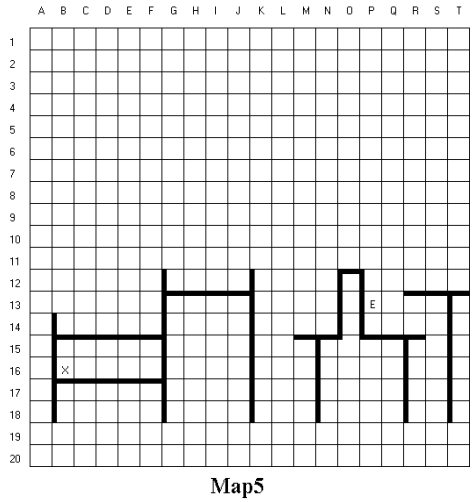

Fig. 5. Showing map5. Bashers, blockers, bombers, builders and climbers are available.

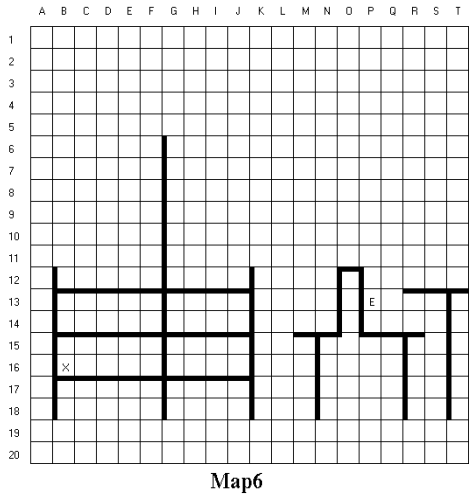

Fig. 6. Showing map6. Bashers, blockers, bombers, builders, climbers and diggers are available.

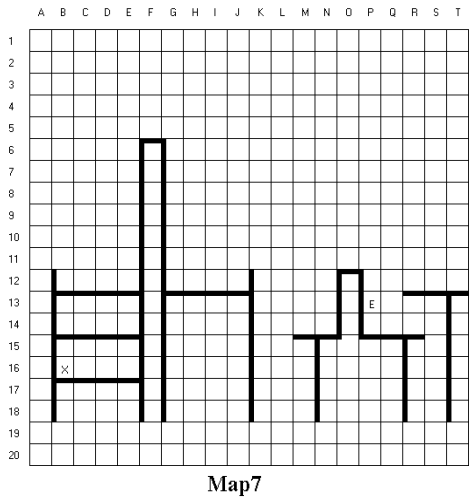

Fig. 7. Showing map7. Bashers, blockers, bombers, builders, climbers, diggers and floaters are available.

by a unique ID. A lemming's ID is its index into the array of lemmings that have entered the map.

The script can be represented by a vector of length *T*. The value *T* is equal to the time-limit imposed by the map. Each component in the vector contains a pair of genes. This pair is a lemming ID, *l*, and a command, *c*. Commands are represented by integers so that

0 = basher 1 = blocker 2 = bomber 3 = builder
4 = climber 5 = digger 6 = floater

The set of alleles for a gene is defined as the set of all of its possible values. The set of alleles for the *l* genes includes an ID for each lemming entering the map, and also the value -1 indicating a null index. For example, if there are 10 lemmings entering the map then the set of alleles for the *l* genes is

$$lSet = \{-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

The set of alleles for the *c* genes includes all the special lemming types available for the map and also the value -1 indicating that the selected lemming should continue with its current command, and the value 256 indicating a nuke of all lemmings. For example, if the blocker and climber special lemming types are available for the map, then the set of alleles for the *c* genes is

$$cSet = \{-1, 256, 1, 4\}$$

The controller examines the script at each time-step *t* of the game and extracts an (l, c) gene pair. The lemming with ID *l* is given command *c*. In this way, there exists a gene pair for all time-steps of a game. For example, a map with a time limit of 5 frames might have the following script

$$Script = (1, 2)\ \ (4, 1)\ \ (2, 5)\ \ (2, 1)\ \ (3, 1)$$

In the first time step, the lemming with ID 1 is ordered to turn into a bomber lemming, in the second time step, the lemming with ID 4 is ordered to turn into a blocker lemming,

and so on. Individuals representing a strategy in this way are placed into a population. Each individual is tested on the test-case map, evaluated and assigned a score. This evaluation takes the form $(\alpha+\beta+\delta+\gamma)/10$. A higher score indicates a better strategy.

$$\alpha = (\textit{numStartingLemmings} * \textit{T}) * \textit{numLemmingsSaved} * 10$$

$$\beta = (\textit{numStartingLemmings} * \textit{timeRemaining}) * 10$$

α rewards every lemming that reaches the exit point. β gives a higher reward if the map is solved in less time. β is set to 0 if the map is not solved. $\alpha + \beta$ is the most intuitive way to reward success. However, it does not reward any lemmings that do not reach the exit.

$$\delta = \sum_{i} (lemming[i].life)$$

δ rewards lemmings that remain active in the map for longer. Here *i* is an index for each lemming that entered the map. δ sums the total life of all lemmings. A lemming's life is incremented once for every time step that it is active in the map. δ is never more than (*numStartingLemmings* * *T*). α is deliberately set so that if a single lemming reaches the exit, then the evaluation of α is more than the evaluation of δ if all of the lemmings were to remain in the map for the duration of the game.

$$\gamma = \sum_{x,y} (explored[x][y]) * 10$$

γ rewards a more diversely explored map with an emphasis on horizontal platform exploration. Here *x* and *y* are the coordinates of the map grid. Each cell in the map initially has an explored score of 0. At the end of the game each cell is scored according to the following

$$explored[x][y] = \begin{cases} 0 & \text{if no lemming entered } cell[x][y] \text{ during the game} \\ 1 & \text{if any lemming entered } cell[x][y] \text{ during the game} \\ 2 & \text{if any lemming entered } cell[x][y] \text{ during the game and if } cell[x][y] \text{ is also a horizontal platform} \end{cases}$$

After evaluation parents are selected from the population with probability according to their score. Offspring are then produced using one-point crossover. For example, the scripts of parent chromosomes p_1 and p_2 produce the scripts of offspring c_1 and c_2.

$$ScriptP_1 = (1,2)\ \ (4,1) \parallel (2,5)\ \ (2,1)\ \ (3,2)$$

$$ScriptP_2 = (7,1)\ \ (5,2) \parallel (8,0)\ \ (1,2)\ \ (4,1)$$

$$ScriptC_1 = (1,2)\ \ (4,1)\ \ (8,0)\ \ (1,2)\ \ (4,1)$$

$$ScriptC_2 = (7,1)\ \ (5,2)\ \ (2,5)\ \ (2,1)\ \ (3,2)$$

Offspring are then mutated using flip mutation. Random gene pairs are replaced with a random selection from the appropriate allele set. For example, the third gene pair of the script for c_1 is mutated.

$ScriptC_1$(pre-mut) = $(1,2)\ \ (4,1)$ **(8,0)** $(1,2)\ \ (4,1)$

$ScriptC_1$(post-mut) = $(1,2)\ \ (4,1)$ **(6,1)** $(1,2)\ \ (4,1)$

It is the authors' hope that the best evolved chromosome will have the optimal number of lemmings being selected for commands. So crossover and mutation are also applied to the set of alleles for the *l* genes. For example

$$lSetP_1 = \{-1, 0, \|2, 6, 8\}$$

$$lSetP_2 = \{-1, 2, \|7, 8, 9\}$$

$$lSetC_1 = \{-1, 0, 7, 8, 9\}$$

$$lSetC_2 = \{-1, 2, 2, 6, 8\}$$

$lSetC_1$(pre-mut) = $\{-1, \mathbf{0}, 7, 8, 9\}$

$lSetC_1$(post-mut) = $\{-1, \mathbf{4}, 7, 8, 9\}$

It should be noted that there is a single, static set of alleles for the *c* genes, that is shared by the entire population of chromosomes. On the other hand, each chromosome has it's own, dynamic, set of alleles for the *l* genes.

Offspring are placed into the population to form the next generation and the process is iterated. The chromosome with the highest score when every generation has evolved is used as the script for the test-case map.

2) Initialisation: Four different initialisation schemes are implemented. TYPEA chromosomes are all initialised to lists of null gene pairs, for example

$$(-1,-1)\ \ (-1,-1)\ \ (-1,-1)\ \ (-1,-1)\ \ (-1,-1)\ \ \ldots$$

TYPEB chromosomes are all initialised to lists of random gene pairs, for example

$$(8,-1)\ \ (4,256)\ \ (-1,4)\ \ (0,-1)\ \ (-1,-1)\ \ \ldots$$

TYPEC and TYPED chromosomes are all initialised in the same way as TYPEA and TYPEB chromosomes respectively, but for their first runs only. For the second, and subsequent, runs TYPEC and TYPED chromosomes are initialised to the best evolved strategy from the previous run. Each run attempts to solve one of the seven test case maps. The best chromosome that solved the previous map is used to initialise the population of chromosomes to solve the next map. It is the authors' hope that strategies learned whilst solving earlier simple maps would be remembered and would help to solve later, more complex, maps.

3) Parameters: Roulette wheel selection is employed with a crossover rate of 0.6 and a mutation rate of 0.001. These values are standard for GAs [24]. The GA was run for 2000 generations. Population size is set to 250 in an attempt to avoid premature convergence of the population [30].

V. Results

All seven test case maps were solved using each of the four initialisation schemes. Figure 8 shows the scores of the best evolved strategies. Closer analysis of the results provides some interesting findings concerning the speed of evolution and the quality of the evolved strategy.

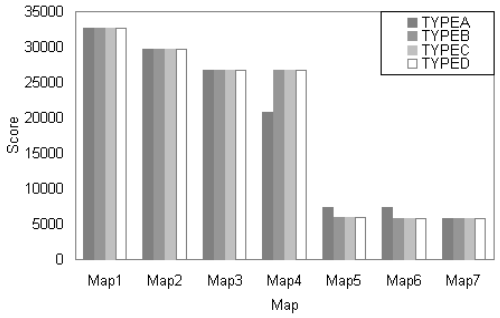

Fig. 8. Showing best scores for each of TYPEA, TYPEB, TYPEC and TYPED initialised chromosomes evolved to solve each of the seven test case maps.

A. Speed of evolution vs. quality of evolved strategy

Figures 9 to 15 show the generation number against evaluation of the best evolved strategies. The bold markers on the y-axis of each graph depicts the highest score achieved by the authors when playing the game.

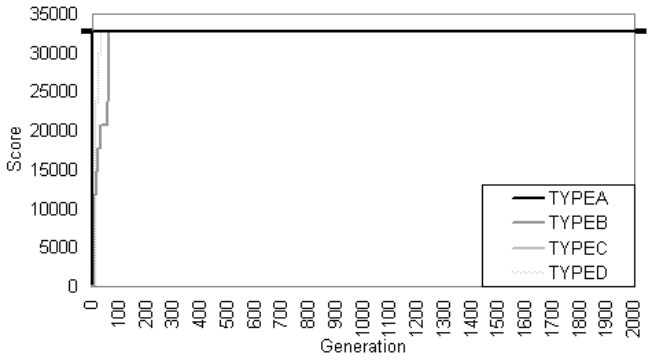

Fig. 9. Showing generations vs. score for map1 run.

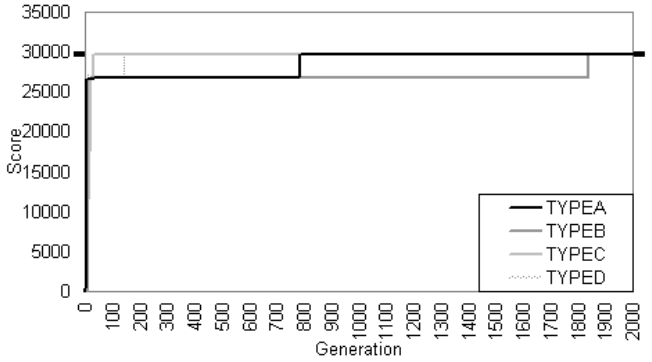

Fig. 10. Showing generations vs. score for map2 run.

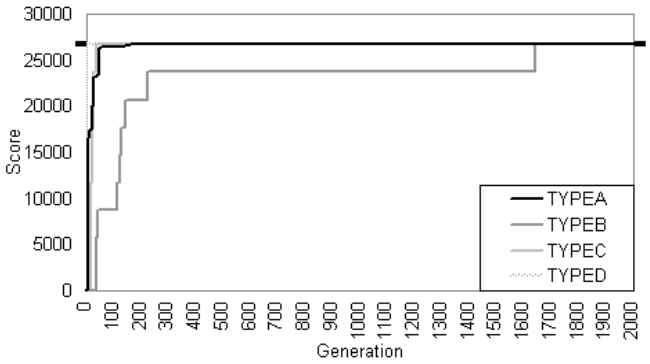

Fig. 11. Showing generations vs. score for map3 run.

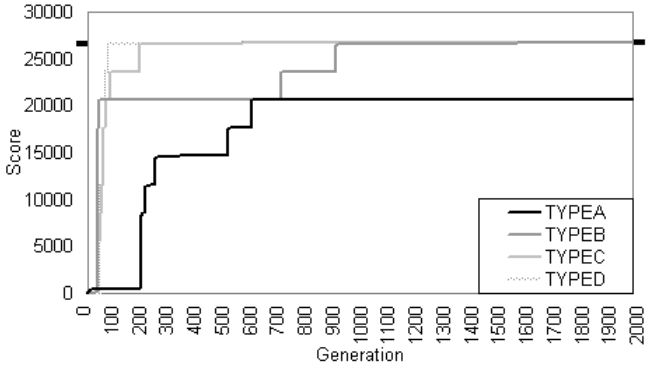

Fig. 12. Showing generations vs. score for map4 run.

TYPEC and TYPED chromosomes generally evolved to a successful strategy faster than the other chromosomes. TYPEB chromosomes typically evolved to a strategy slower than the other chromosomes. Figure 10 and figure 14 show TYPEC chromosomes evolving to a successful strategy 1812 and 1822 generations quicker than TYPEB chromosomes respectively.

TYPEB, TYPEC and TYPED chromosomes consistently evolved to equally high scores. Figure 8 shows TYPEA chromosomes evolving to a strategy to solve map4 with a score visibly lower than the strategies emerging from the other chromosomes. Figure 8 also shows TYPEA chromosomes evolving to a strategy to solve map5 and map6 with a score visibly higher than the strategies emerging from the other chromosomes. In fact, figure 13 shows a strategy emerging from TYPEA chromosomes that is better than the strategy that was formed by the authors.

In summary, populations initialised with TYPEC or TYPED chromosomes are more likely to evolve to a successful strategy more quickly, whereas populations initialised with TYPEA chromosomes will offer a greater diversity in the final score. To understand why these patterns emerge we analysed the evolved scripts used to solve the maps.

1) Speed of evolution: TYPEB chromosomes consistently evolved the slowest. This is intuitive to understand. To play *Lemmings* well, it is often better to do nothing and allow events to unfold automatically. To solve map7 it is necessary to apply the correct commands at only 18 time-steps and remain

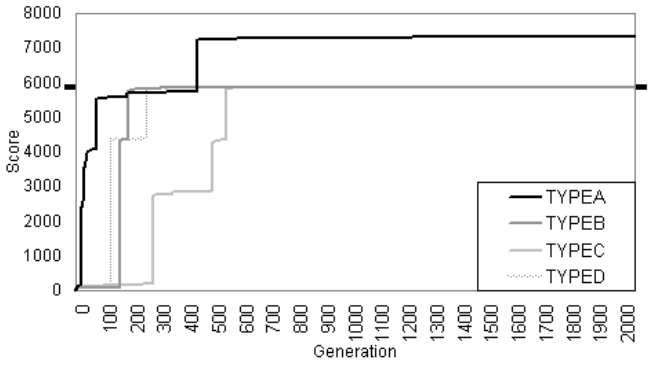

Fig. 13. Showing generations vs. score for map5 run.

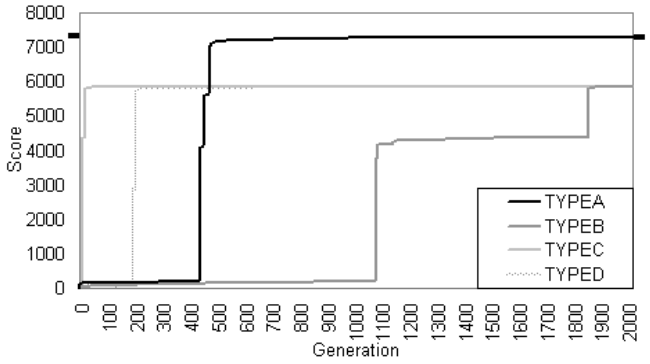

Fig. 14. Showing generations vs. score for map6 run.

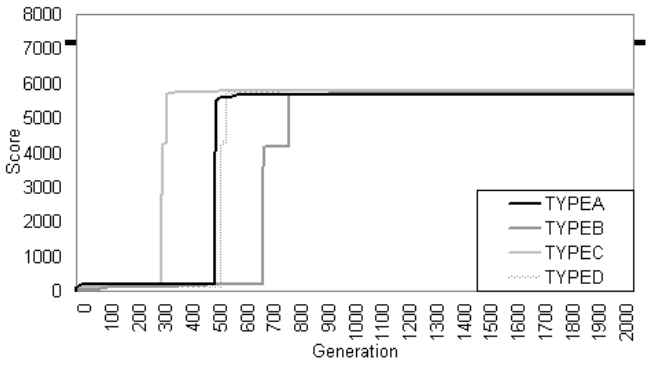

Fig. 15. Showing generations vs. score for map7 run.

	Map5		Map6		Map7	
	lemming	command	lemming	command	lemming	command
1	0	-1	2	5	2	5
2	0	-1	0	3	4	5
3	2	0	0	-1	3	5
4	0	1	0	1	0	1
5	0	0	4	0	4	0
6	4	3	2	4	3	1
7	0	0	3	4	2	4
8	2	4	0	4	4	-1
9	4	4	0	4	0	3
10	2	-1	0	5	0	4
11	1	-1	0	4	1	-1
12	0	0	4	-1	1	6
13	3	4	3	4	3	4
14	0	-1	2	5	3	3
15	4	4	4	4	4	4
16	1	4	1	4	1	4
17	2	3	2	3	2	3

Fig. 16. Showing (l, c) gene pairs for the first 17 time steps of map5, map6 and map7. (l, c) pairs responsible for a change in lemming state are highlighted, all other (l, c) pairs were ignored. A pair is ignored if either the lemming *l* does not exist, or if the command *c* cannot possibly be completed.

inactive for the other 282 possible time-steps. A successful strategy will therefore consist mostly of NULL commands. Chromosomes initialised as lists of random gene pairs will most likely be at a disadvantage. The GA is effectively spending time NULLifying gene pairs that, at least in other initialisation schemes, might be initialised to NULL in the first place.

TYPEC and TYPED chromosomes evolved the quickest. It is less trivial to explain this phenomenon. We analysed map5, map6 and map7 and the best evolved strategy that emerged from the TYPEC chromosomes to solve these maps. The first sections of map5, map6 and map7 are identical (see Figures 5 to 7). The lemmings drop from the entrance at grid co-ordinate (P,13). The first lemming needs to block before falling off the platform at (S,14). The remaining lemmings need to be turned into climbers so that they make it over the small turret at (P,14). The first one over the small turret needs to build a bridge over the gap at (M,14) and up to the platform in the middle of the map at (J,12). Figure 16 shows the commands that change the lemmings' states to navigate them to (J,12). There are six commands for map5. Five of the six commands are remembered and used for map6. All five of these commands from map6 are remembered and used for map7. The strategy to solve map7 consists of 18 commands in total. In this case nearly a third of the required commands were inherited. A chromosome with such a head-start is likely to require less time to evolve.

2) Quality of evolved strategy: TYPEA chromosomes were the only ones to offer diversity in their final score. We again analysed map5, map6 and map7. Figure 13 shows a sub-optimal strategy resulting from TYPEC chromosomes. TYPEC and TYPED chromosomes inherit bad genes along with good ones. Consequently, figure 14 and figure 15 show sub-optimal solutions for map6 and map7 resulting from TYPEC chromosomes. TYPEA chromosomes do not inherit genes. Figure 12 shows a solution resulting from TYPEA chromosomes that is worse than the other solutions for that map. Figure 13 shows a solution resulting from TYPEA chromosomes that is better than the other solutions for that map. TYPEA chromosomes are not effected by the success or failure of previous runs and are more likely to offer diversity.

B. General strategy

Ideally, a strategy should be general enough to solve multiple maps. We tested the best evolved strategies from TYPEC and TYPED chromosomes that were able to solve map7, against map1 through map6 to see if they displayed any backward-compatibility. The best evolved strategy from TYPEC chromosomes was able to solve map1 along with map7. The best evolved strategy from TYPED chromosomes was able to solve map2 along with map7. Each strategy successfully solved its test-case map, but was not general enough to solve other maps.

VI. Future Work

There are two main issues for future work. We aim to improve the representation of *Lemmings* maps so that a more general strategy can be evolved. We also aim to co-evolve the *Lemmings* maps against the strategy that solves them.

A. Representation

The representation described in this paper breaks the *Lemmings* map into nothing more than time-steps. Generalisation is not achievable with such a basic representation. A strategy that uses this representation relies on a blind sampling and does not scale well. We would like to improve the representation of *Lemmings* maps so that a more general strategy can be evolved. The geography of the *Lemmings* maps and the lemmings needs to be encoded. This representation can then be applied to a strategy evolved using an approach such as genetic programming.

The Genetic Programming (GP) [31] paradigm is similar to a genetic algorithm. In GP each chromosome in the population forms a computer program. John Koza [31] used GP to successfully evolve S-expressions to navigate an ant along a trail of food. Luke et al [32] successfully applied GP to behaviour-based team coordination for RoboCup Soccer.

It is the authors' intention to apply GP to *Lemmings* so that a more general, scalable, strategy can be evolved. The strategy should at least be backwards-compatible for the maps it has previously solved. Once a scalable strategy is obtained, it is the authors' intention to gradually scale-up the complexity of the game until it matches that of the original *Lemmings* game.

B. Co-evolution: environment vs. strategy

Chellapilla and Fogel [13], [14] and Fogel [15] successfully co-evolved a program to play a game of checkers. Members of the population played against each other to compete for survival into the next generation.

Further research will aim to automate the process of map creation so that there is co-evolution between the *Lemmings* maps and the strategy that solves them. This might lead to an open ended game where a more complex strategy evolves as the maps evolve in complexity.

VII. Conclusion

An evolutionary approach is a promising direction for developing a computer program to play the game of *Lemmings*. We present initial work, where scripts are evolved to successfully navigate lemmings through increasingly difficult maps. Populations that inherited genes from the best evolved strategy of the previous run evolved in the shortest time. Populations that were initialised for every run as NULL commands offered more diversity in their final score.

References

[1] McCarthy J. Partial formalizations and the lemmings game. Technical report, Stanford University, Formal Reasoning Group, 1995.

[2] Campbell M, Joseph Hoane Jr A, and Hsu F-h. Deep blue. *Artificial Intelligence*, 134:57–83, 2002. Also in Schaeffer J and Jaap van den Herik H, editors, *Chips challenging champions*, pages 97-123. Elsevier Science, 2001.

[3] Turing A, Strachey C, Bates M, and Bowden B. Digital computers applied to games. In Bowden B, editor, *Faster than thought*, pages 286–310. Pitman, 1953.

[4] Shannon C. Programming a computer for playing chess. *Philosophical Magazine*, 41(4):256–275, 1950.

[5] Berliner H and Ebeling C. Hitech. In Marsland T and Schaeffer J, editors, *Computers, chess and cognition*, pages 79–109. Springer-Verlag, 1990.

[6] Condon J and Thompson K. Belle. In Frey P, editor, *Chess skill in man and Machine*, pages 201–210. Springer-Verlag, 1982.

[7] Hyatt R, Gower A, and Nelson H. Cray blitz. In Marsland T and Schaeffer J, editors, *Computers, chess and cognition*, pages 111–130. Springer-Verlag, 1990.

[8] Hsu F-h, Anantharaman T, Campbell M, and Nowatzyk A. Deep thought. In Marsland T and Schaeffer J, editors, *Computers, chess and cognition*, pages 55–78. Springer-Verlag, 1990.

[9] Samuel A. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3:211–229, 1959.

[10] Samuel A. Some studies in machine learning using the game of checkers - recent progress. *IBM Journal of research and development*, 11:601–617, 1967.

[11] Schaeffer J, Culberson J, Treloar N, Knight B, Lu P, and Szafron D. A world championship caliber checkers program. *Artificial Intelligence*, 53:273–289, 1992.

[12] Schaeffer J. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer-Verlag, 1997.

[13] Chellapilla K and Fogel D. Evolution, neural networks, games and intelligence. *Proc. IEEE*, 87(9):1471–1496, 1999.

[14] Chellapilla K and Fogel D. Evolving neural networks to play checkers without relying on expert knowledge. *IEEE Trans. Neural Networks*, 10(6):1382–1391, 1999.

[15] Fogel D. *Blondie24 playing at the edge of AI*. Morgan Kaufmann, 2002.

[16] Müller M. Computer go. *Artificial Intelligence*, 134:145–179, 2002.

[17] Billings D, Burch N, Davidson A, Holte R, Schaeffer J, Schauenberg T, and Szafron D. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of IJCAI-03*, 2003.

[18] Kendall G and Willdig M. An investigation of an adaptive poker player. *Proc of the 14th Australian Joint Conference on Artificial Intelligence*, pages 189–200, 2001.

[19] Tesauro G. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134:181–199, 2002.

[20] Moriarty D and Miikkulainen R. Discovering complex othello strategies through evolutionary neural networks. *Connection Science*, 7(3):195–209, 1995.

[21] Sheppard B. World-championship-caliber scrabble. In Schaeffer J and Jaap van den Herik H, editors, *Chips challenging champions*, pages 283–317. Elsevier Science, 2001. Reprinted from Artificial Intelligence 134 (2002) 241-275.

[22] Schaeffer J. The games computers (and people) play. In *AAAI/IAAI*, page 1179, 2000.

[23] Jaap van den Herik H, Jos W Uiterwijk, and Jack van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134:277–311, 2002.

[24] Beasley D, Bull D, and Martin R. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.

[25] Goldberg D. *Genetic Algorithms in search, Optimization, and Machine Learning*. Addison Wesley, 1989.

[26] Holland J. *Adaption in Natural and Artificial Systems*. MIT Press, 1975.

[27] Man K, Tang K, and Kwong S. *Genetic Algorithms*. Springer, 1999.

[28] Alliot J and Durand N. A genetic algorithm to improve an othello program. *Artificial Evolution*, pages 307–319, 1995.

[29] Donelly P. Evolving go playing strategy in neural networks, 1994. AISB Workshop on Evolutionary Computing.

[30] Leung Y, Gao Y, and Xu Z. Degree of population diversity - a perspective on premature convergence in genetic algorithms and its markov chain analysis. *IEEE Transactions on Neural Networks*, 8(5):1165–1171, 1997.

[31] Koza J. *Genetic Programming: On the programming of computers by means of Natural Selection*, chapter Four introductory examples of Genetic Programming, pages 147–162. The MIT Press, Cambridge, Massachusetts, 1992.

[32] Luke S, Hohn C, Farris J, Jackson G, and Hendler J. Co-evolving soccer softbot team coordination with genetic programming. *RoboCup-97: Robot Soccer World Cup I (Lecture Notes in Artificial Intelligence No. 1395)*, pages 398–411, 1997.