POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

EXECUTIVE SUMMARY OF THE THESIS

Extension of the ADAPTA architecture applied to the videogame Advance Wars

TESI MAGISTRALE IN COMPUTER SCIENCE AND ENGINEERING – INGEGNERIA INFORMATICA

AUTHOR: LORENZO CARNAGHI

ADVISOR: DANIELE LOIACONO

ACADEMIC YEAR: 2020-2021

1. Introduction

In 2008, Maurice Bergsma and Pieter Spronck proposed an AI architecture, named ADAPTA [1], for a deeply simplified version of the Turn Based Strategy (TBS) videogame Advance Wars™. The aim of this work is to extend the proposed concepts to the actual game and lay the foundation for possible future researches on the topic of AI for Turn Based Strategy (TBS) videogames, which features a general scarce literature.
The project is built on top of an open-source version of the game, called Commander Wars [2].

2. ADAPTA overview

The purpose of the ADAPTA architecture is to create an AI which is able to adapt to the player and change its strategy across different matches. Its aim is not necessarily to win, but instead is to create an interesting experience for the player to keep them in the flow [3].
The architecture is structured modularly: it is divided into several Tactical Modules and a Strategic Module. Each Tactical Module has its own behavior and controls in general a subset of units. Tactical Modules are divided into Adapta Modules and Building Modules: the firsts control units, the seconds decide which units should be built. Each Tactical Module proposes a bid for each unit, ranging from 0 to 1, indicating the usefulness of that unit for their own tactic.
A Strategic Module controls the Tactical Modules by assigning them a weight at the start of the match to adjust its strategy.

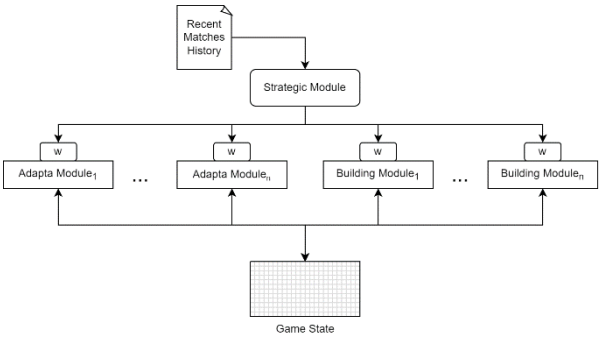


Figure 2-a ADAPTA modules overview

Since the modules must share a limited number of assets (units), the Adapta AI (which is the AI built with the ADAPTA architecture) receives the bids

of the modules and then assigns each unit to the modules which had overall the highest, weighted, bid.

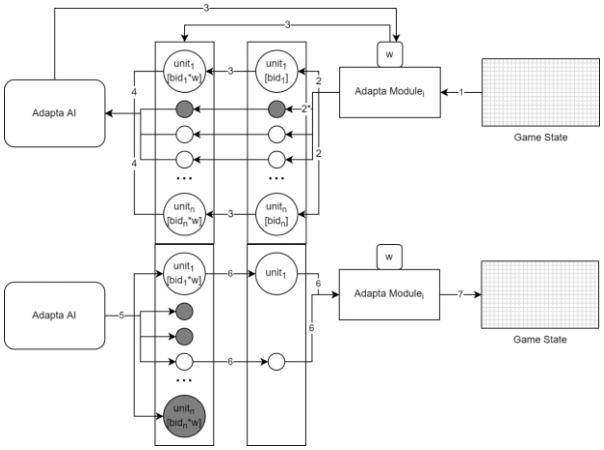
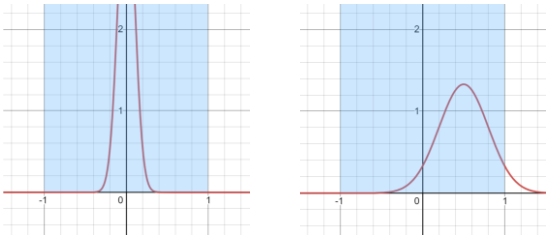


Figure 2-b unit assignment and processing of modules

Each module can be responsible for any subcategory of units or actions in general. In theory all the modules could be purely scripted and behavioral, each accounting a specific subtask. In this scenario the ADAPTA would behave in a similar way to Dynamic Scripting [4].
Given a specific configuration of modules, the Strategic Module tracks a history of past matches and uses it to assign the weights of the modules for the next match, to change the overall strategy adopted.
First it evaluates the *historic value* of each module, which is calculated as follows:

$$historic\_value(m) = \frac{\sum_{h=0}^{H} w_{m,h} * r_h * p^h}{\sum_{h=0}^{H} p^h}$$

Where h is the index of the past match (0 is the most recent, H is the history capacity), $w_{m,h}$ is the weight assigned to module m at past match h, r_h is the outcome of match h, p^h is the past multiplier, a value in range [0,1] which gives more or less importance to past matches. This allows to evaluate, based on previous matches, the contribution of each module to the overall victory or loss of the module. This is not per se a value which promotes better performing modules, because the aim of the Strategic Module is not necessarily to win. This historic value, which ranges from -1 to 1, is passed as argument in a simple gaussian, which has mean μ representing the *aim* of the Strategic Module. If is set to 0, it will try to change its configuration in order to achieve a tie with the opponent. If the past configurations for instance have brought to a solid win, the modules with more responsibility for that will be penalized.



(a) $\mu = 0, \sigma^2 = 0.1$ tries to tie, spiked weights distribution

(b) $\mu = 0.5, \sigma^2 = 0.3$ Tries to slightly win, smoother distribution

Figure 2-c Strategic Module configuration examples

In the above picture, historic values are placed on the x in the range [-1,1], while the final weight is the gaussian value in that point.
To avoid converging to uniform distribution of weights, the Strategic Module can be configured to have a degree of noise, both in single weights and both a chance of extracting a fully random configuration.

3. MIN Module

The Multi Influence Network Module, or MIN for short, is the extension of the one implemented in the original paper [1], applied to the real game.
It is a module which is dedicated to the Extermination concept of the 4 X's (Exploration, Expansion, Exploitation, and Extermination).
The core logic is the computation of the optimal tile for a unit through the use of Influence Maps [5]. An influence map accounts and propagates a weight relative to each element considered relevant, according to a propagation function.
The general formula for evaluating the influence on a tile is:

$$I(x,y) = \sum_{o}^{O} p(w(o), \delta(o,x,y))$$

Where O is the set of the objects accounted by the map, $p(W,d)$ is a propagation function of a weight vector W and a distance d, $w(o)$ is a function which converts the object into a vector of weights, and $\delta(o,x,y)$ is a distance function which evaluates the distance in game tiles of object o from the tile with coordinates (x,y).
The set of objects accounted by the specific map depends on the map chosen. For the MIN, several types of maps were created: there is a set of 3

learnable maps which propagates learned weights for allied units, enemies, or both; a set of 2 maps which propagates values based on damage dealt and taken according to the game's damage chart, and a map which assigns an influence based on each tile on the map. A MIN Module can be configured to have any number of this maps, even replicated, since in any case the weights assigned to each maps are learned and could bring different meanings to different units.
Since there are different types of units in Advance Wars, some information may be shared. The MIN Module hence has a set of local (l) and global (g) maps: local maps are relative to each type of unit, while global are computed for the whole module.
The final influence of the output map for each type of unit is computed as such:

$$I(x,y) = \sum_{l}^{L} i_{l(x,y)} * w_l + \sum_{g}^{G} i_{g(x,y)} * w_g$$

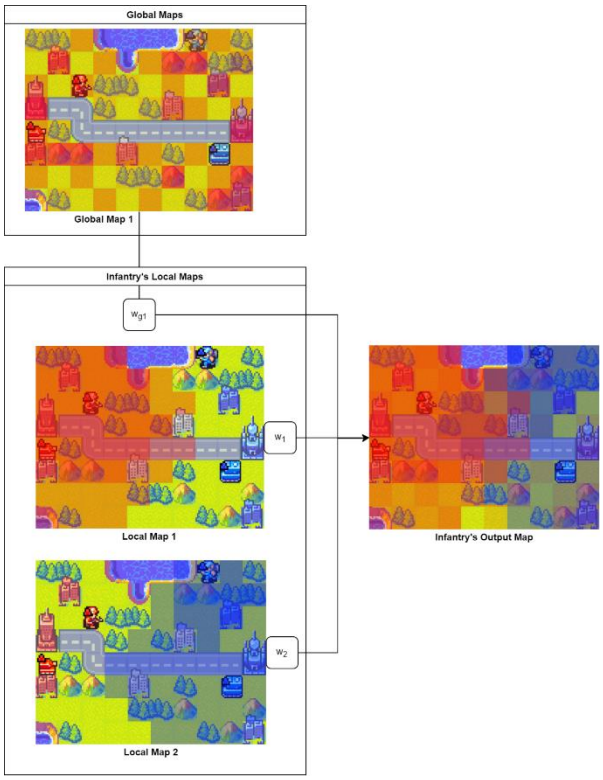


Figure 3-a MIN Module output computation

Which is a weighted sum of every local and global map. Even if the map is global, the weight used by a type of unit (e.g., Infantry) is still local, because it can exploit in a different way global information.
A MIN Module is configured with any number and types of local and global maps (although some maps can only be local or global). Local maps are replicated once per supported type of units, global maps are computed once. Since they are computed at the start of turn, before moving a unit its influence map is updated by adjusting the influence of every element accounted that has changed since last computation. For performance reason this doesn't recompute the map and the operation is performed only when needed.
The propagation of unit influences is performed stepwise, which means that tiles reachable or attackable by a unit in the same number of turns will have the same influence.

Evolution is performed by simply converting the weights which define a MIN Module's maps into a vector of weights, and performing a classical genetic algorithm approach. It was adopted a custom fitness function to slightly boost better offspring and penalize worse ones:

$$f'_{cust} = (f + f_{max} - f_{min})^4$$

Where f_{max} and f_{min} are the boundaries of the obtainable fitness.
It was also included the option to perform a form of Transfer Learning, which allows to transfer all the possible information from a MIN Module to a new one with a different configuration. This allows to train a MIN Module in a more contained environment and then to pass its knowledge to a new module which is trained in a different environment with a superset of units and/or in a different maps, starting with a set of non-random weights associated to the units used by the previous module already trained. The transferred weights can be relearned but can be used as a more stable basis of weights w.r.t. a starting random one.

4. CNN Module

The CNN Module is an experimental approach which makes use of Convolutional Neural Networks for evaluating and processing a game state map.
The CNN is adapted from a segmentation problem: the input is the game map and the output is still a representation of the game map. The idea is to read gradually local points of the map and building a state which represents the knowledge of the current game, to be then applied to the map. Compared to classical image segmentation problem, Advance Wars features a strongly smaller sized input, however each tile carries way more information than a pixel and is hence deeper. A tile is in fact fully defined by its terrain and its

unit, and each one has a set of unique feature to account for. To summarize them and reduce complexity by keeping only the more meaningful, a CNN Module can include a customized set of features both for terrains and units.
For terrains is defense value, movement type, owner, and a static feature.
For units is move points, the ability to capture, movement type, ammos, and a static feature.
Some features (movement type and static features) can be replicated more than once with different meanings, making the input deeper. Static feature are heuristics that can be defined in configuration, to summarize some concept (like, for units, goodness against vehicles in general) and reduce input size. Additionally units' static features s_f are multiplied by the owner (-1 if opponent, +1 ally) and their health (range [0,10]) to account for their state in game:

$$s'_f = s_f * owner(u) * hp\_ratio(u)$$

The game map sized $w * h$ is hence adapted to an input map sized $w * h * d$ where d is the number of total features.
The output map is instead deep as the number of units supported by the CNN Module. Each slice of the output map is the final map relative to a type of unit, which indicates the desirability of acting on that tile. The CNN Module has also the ability to make infantry units capture buildings.

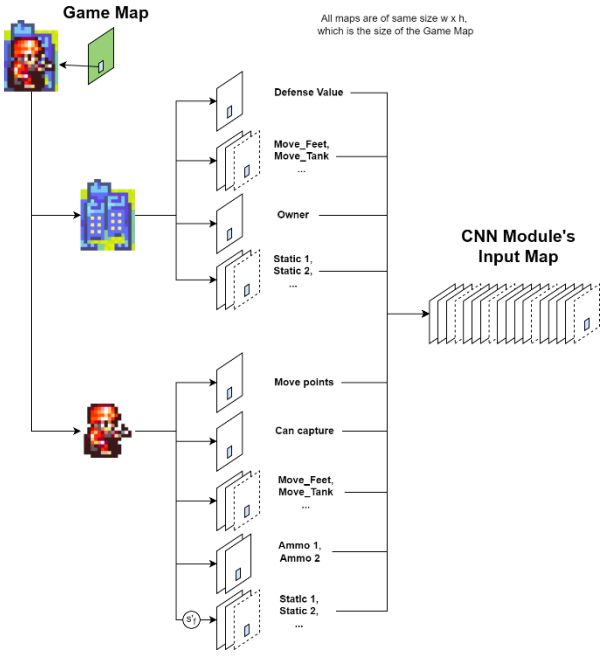


Figure 4-a Computation of CNN Module's input map

The evolution of the CNN is performed through NEAT [6], in particular an application which uses Compositional Pattern Producing Networks (CPPN), called HyperNEAT [7]. The idea is to evolve networks which from the simplest structure become more complex over time in different ways; these networks act like functions in a N-dimensional space, creating patterns.

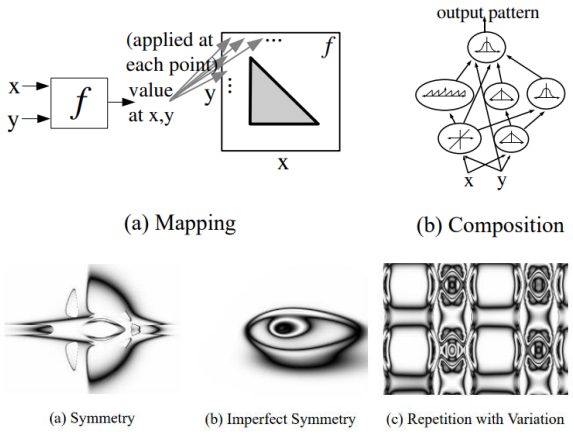


Figure 4-b CPPN encoding and patterns (from [7])

In this context, CNN kernels of the network are disposed in a 3D space (Figure 4-c), with parametrized distances. The CNN itself is hence fixed once configured, but the weights are learned through the use of this technique.

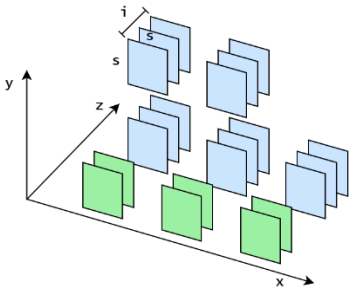


Figure 4-c CNN disposition in 3D space

It is still possible to evolve the CNN Module through classic genetic algorithms.

5. Experiments and results

5.1. Training environment

The training process of every Adapta Module consists in a series of matches made in ad hoc custom maps. Each one makes use of the basic elements of the game in different ways to tackle different aspects. All maps are on the small side of the spectrum of Advance Wars maps, since bigger maps do not necessarily demonstrate the potential

of a method, but complexity and/or training time explodes as the map grows in size.

Table 5-1 Complexity of a small Advance Wars Map and other board games [8]

Game	Possible configurations	Board size
Checkers	$5 * 10^{20}$	*8x8*
Chess	$8.7 * 10^{45} (upper\ bound)$	*8x8*
Shogi	$\cong 10^{71}$	*9x9*
Go	$\cong 2.089 * 10^{170}$	*19x19*
Advance Wars	$\cong 10^{95} (raw\ estimate)$	*8x8*

All the matches are **pre-deployed**, for simplicity of training and evaluation. The opponents faced are the ones already built in the open-source project, which are named Very Easy AI and Normal AI, which has 3 configurations: normal, offensive, defensive.

(a) – Training map 1

(b) – Training map 2

(c) – Training map 3

(d) – Training map 4

Figure 5-a training maps

The trainee can be trained in any number of maps, but in any case is placed in both sides because the slight asymmetry of the maps can force a different playstyle.
Also since there is some degree of randomness in the training, namely because of the luck mechanic in the game (which is a very minor variable most of the times, at most increases by 10% the damage) and because of the behavior of the opponent AI, the training is performed multiple times on the same map. At the cost of increasing training time, this increases stability of solutions and selective pressure.
Since the experiments were done in pre-deployed maps, the fitness is evaluated mainly accounting for the **army strength ratio** (a_r), evaluated as:

$$a_r = \frac{A_{last\_turn}}{A_{first\_turn}}$$

Where A_t is the **value** of the army at a certain turn t, defined as:

$$A_t = \sum_{u}^{U} val(u) * hp\_ratio(u)$$

U is the set of units composing the army, $val(u)$ is the unique value in War Funds (the in-match currency of the game) of the unit, $hp\_ratio(u)$ is the percentage of HP left to unit u.
a_r evaluates the percentage of strength left to a predeployed army.
For MIN Modules fitness is evaluated returning a_r^{ally} in case of victory and $-a_r^{opponent}$ in case of loss. For multiple matches the partial fitnesses of each match are summed up. Has range [-1, 1], for N matches [-N, +N].
For CNN Module it is evaluated a variant which sums 1 to positivize the fitness, averages the partial fitnesses and *after* averaging sums +1 for each victory obtained. Has range [0, 2] for each match and [0, 2+N] for N matches.

5.2. Experiments

Meaningful experiments to be mentioned are:

MIN Module 1 and 2
MIN Module 1 was trained in map 2, which is the one featuring the simplest complexity in terms of units. It obtained a solid win margin and managed to stabilize also average fitnesses above 0.
Module 2, trained on map 3 which features a bit more variety in terms of unit types, still managed to find a good solution but not as dominant, highlighting the difference in results with a relative increase in search space, and both proving the potential of the MIN Module.

Module 3 and 4
These two modules were trained both on map 1, which is the biggest and with the highest number of units, with the same configuration. However Module 3 started with previous knowledge transferred by MIN Module 1. Although the best absolute result was obtained by Module 4, the average fitnesses graph shows the improvement given by the transfer learning.

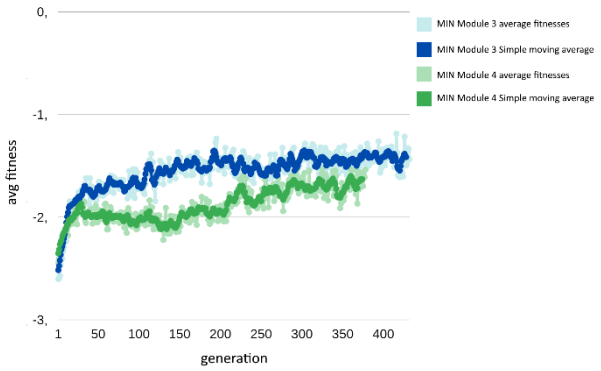


Figure 5-b Average fitness of MIN Module 3 and 4 by generation

CNN Modules 1, 2, 3

CNN Module 1 is set up as the most complex of the 3 in terms of CNN configuration, and didn't manage to win against the Very Easy AI.

CNN Module 2 is trained in the same map (3) but is simpler, and managed to win a match, still obtaining an overall loss, however. The same is true for CNN Module 3, which was trained in map 2. Both shown however a better average performance, may indicating that simpler models suits better for contained environments, or the models simply managed to find a better solution because of the search space more limited. Also having managed to win shows that the CNN Module has the potential to bring interesting behaviors, but its training is more challenging.

Strategic Module

As a final experiment, a Strategic Module was made playing against several different AIs in map Training 2.

Its aim was set to 0, so that it tries to change strategy in order to tie.

First 12 matches were against the Normal AI, then 12 matches against Very Easy, and then Offensive Normal AI. After that the cycle repeats.

The Adapta AI assigned was configured to have 2 behavioral modules (offensive and defensive), 2 MIN Modules and 2 CNN Modules.

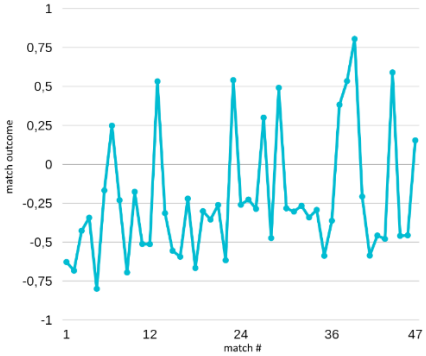


Figure 5-c Strategic Module results

Considering that this module keeps memory only of the last 16 matches, it is not really necessary to test it on long runs of matches because there's no learning and the overall trend is constant.

The results show that the Strategic Module's change in weight configurations brought a different matches outcome, both with the same and with different AIs, which is in line with the aim of the overall ADAPTA architecture. Similar experiments in other maps obtained comparable results.

6. Conclusions

The results obtained were in line with the scope of the work.

The MIN Module has proven to be capable of achieving interesting and valid behaviors.

The CNN Modules didn't achieve notable results but the ones obtained can still suggest that it can be a valid approach.

This works could be utilized a foundation for future researches, both applied to this same game, tackling different problems or the same ones in different ways, or to other TBS games, given that the strong point of this architecture is its customizability.

7. Bibliography

[1] M. Bergsma and P. Spronck, "Adaptive Spatial Reasoning for Turn-based Strategy Games," Tilburg centre for Creative Computing, Tilburg University, The Netherlands, Tilburg, 2008.

[2] R. Muller, "CommanderWars project: https://github.com/Robosturm/Commander_Wars".

[3] J. Schell, "The Experience is in the Player's Mind - Focus," in *The Art of Game Design: A Book of Lenses*, Burlington, Morgan Kaufmann Publishers (Elsevier), 2008, pp. 118-123.

[4] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper and E. Postma, "Adaptive game AI with dynamic scripting," Springer Science + Business Media, LLC, 2006.

[5] P. Tozour, "Using a Spatial Database for Runtime Spatial Analysis," in *AI Programming Wisdom 2*, Charles River Media, 2004, pp. 381-390.

[6] K. O. Stanley, "Evolving Neural Networks through Augmenting topologies," The MIT Press Journals, Austin, 2002.

[7] K. O. Stanley, J. Gauci and D. D'Ambrosio, "A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks," Orlando, Florida, 2009.

[8] "Game Complexity - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Game_complexity.