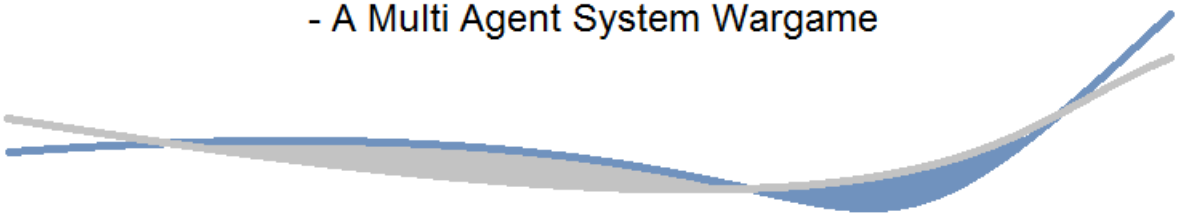
Compiler and Language Development

- A Multi Agent System Wargame



Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Øst
Telephone +45 9940 9940
Telefax +45 9940 9798
http://cs.aau.dk

Title: Wargame

Subject: Language engineering

Semester: Spring Semester 2011

Project group: sw402a

Participants:
Henrik Klarup
Kasper Møller Andersen
Kristian Kolding Foged-Ladefoged
Lasse Rørbæk
Rasmus Aaen
Simon Frandsen

Supervisor:
Jorge Pablo Cordero Hernandez

Paolo Viappiani

Number of copies: 9

Number of pages: MISSING

Number of appendices: 3

Completed: 27. May 2011

Synopsis:

In this project, an agent oriented language is designed and implemented. The implementation is done via a high-level to high-level compiler. The language is specialized towards a concept we call "multi agent wargame". This wargame gives the user the possibility to simulate programmed battle scenarios.
The language is designed using BNF and EBNF grammar, and implemented via abstract syntax trees and tree traversal. The implementation is described through a big step semantic.
Furthermore we discuss the different aspects of the language and ways to improve it and then compare it to an object oriented language to determine the up- and downsides of this kind of specialized language.
We arrive at the conclusion that the language does exactly what it is supposed to do; provide programmers with a simple language to express a battle scenario.

Preface

This report is written in the fourth semester of the software engineering study at Aalborg University, spring 2011.

The goal of this project is to acquire knowledge about fundamental principles of programming languages and techniques to describe and translate programming languages in general. Another goal is to get a basic knowledge of central computer science and software technical subjects with a focus on language processing theories and techniques.

We will achieve these goals by designing and implementing a language optimized for controlling a multi agent system in the form of a wargame, which we call *MASSIVE* - **M**ulti **A**gent **S**imulation**S**ystem **I**n **V**irtual **E**nvironment. The product have been written entirely in C# using Visual Studio, this have been done because Visual Studio have a great framework to help develop interfaces. Since its easy to create interfaces with the Visual Studio framework, there have been more time to create the more important part of the project.

Source code examples in the report is represented as follows:

```
1 if (spelling.ToLower().Equals(spellings[i]))
2   {
3     this.kind = i;
4     break;
5   }
```

Source code 1: This is a sorce code example

We expect the reader to have basic knowledge about object oriented programming and the C# language.

Contents

Part I

Discussion

In this part we discuss our project. We descripe a use case of the MASSIVE language and from that we show how our language has lived up to its purpose. Also we list some advantages and disadvantages of the MASSIVE language versus other object oriented programming languages, i.e. C#, and finally we conclude on the project as a whole.

CHAPTER 1

Language Development

1.1 Compiler language

We decided early on to develop our compiler in C#, because it is a language we have a lot of experience with, and the object oriented paradigm is helpful in developing a compiler that uses an abstract syntax tree. There were problems managing reference types in C# though. Reference types are the kinds of objects that when created refer to an existing object in memory rather than creating a new instance of the object.
Several bugs occurred due to difficulty in anticipating when something is a referenced type as opposed to a seperate object.
It might therefor have been beneficial to develop the compiler in a language like Haskell, which uses the functional paradigm. This is because purely functional languages do not allow side effects in their functions, meaning that existing data is not altered. Haskell is one such language [**?**], where new data is created and the alterations are applied to, so reference types are of no concern.

CHAPTER 2

MASSIVE Language

In this report we illustrates how we designed and implemented the agent oriented language, MASSIVE. During this chapter we demonstrates a working simulation with a use case, and compare the agent oriented language to Object Oriented Code (C#). Furthermore we discuss the advantages and disadvantages of the MASSIVE language.

2.1 Use Case

In this use case we demonstrate how to write a mini-game in our language, how to compile it and how to play it.

The first thing one needs to do is to write some MASSIVE code. In the following code example are examples of MASSIVE code, however there are features of the langauge that are not being used in this example. For a full code referernce please check **??**. In the example two teams are created called "Disco" and "Kman", agents are added to them and at the end a simple action pattern is defined, later to be used when running the simulation.

```
1
2  /* Initializes the game */
3  Main ()
4  {
5
6    // Creates team Disco.
7    new team teamDisco("Disco", "#FF6600");
```

```
8   num totalDiscos = 10;
9   for ( num i = 0; i < totalDiscos; i = i + 1)
10  {
11    num a = 0;
12    if ( i < totalDiscos-1 )
13    {
14      a = 1;
15    }
16    else
17    {
18      a = 21-totalDiscos;
19    }
20
21    new Agent newAgent("Stue", a);
22    teamDisco.add(newAgent);
23  }
24
25  new team teamKman("Kman", "#660000");
26  new squad squadNabs("noobs");
27  new squad squadRevo("Revolution");
28
29  for(num i = 0; i < 4; i = i + 1)
30  {
31    num a = 0;
32    if(i =< 1)
33    {
34      a = 2;
35    }
36    if(i >= 2)
37    {
38      a = 8;
39    }
40
41    new Agent newAgent("Kman", a);
42    teamKman.add(newAgent);
43
44    if (i <= 1)
45    {
46      squadNabs.add(newAgent);
47    }
48    if (i => 2)
49    {
50      squadRevo.add(newAgent);
51    }
52  }
53
54  // Moves used in the actionPatterns.
55  string moveUp = "unit move up";
56  string moveDown = "unit move down";
```

```
57    string moveLeft = "unit move left";
58    string moveRight = "unit move right";
59
60    // Creates the action pattern Patrol Low.
61    // Patrols the lower part of the game area.
62    new actionpattern patrolLow("PatrolLow");
63    patrolLow.add(moveUp);
64    patrolLow.add("unit move 25,24");
65    patrolLow.add(moveUp);
66    patrolLow.add("unit move 0,23");
67    patrolLow.add(moveDown);
68
69 }
```

Source code 2.1: MASSIVE code example

When compiling this code the compiler warns that there are unused variable (see **??**). We will disregard this for the purpose of this use case, however, if there were serious faults in the code the compiler would warn you the same manner and maybe even refuse to compile if the faults were serious enough.

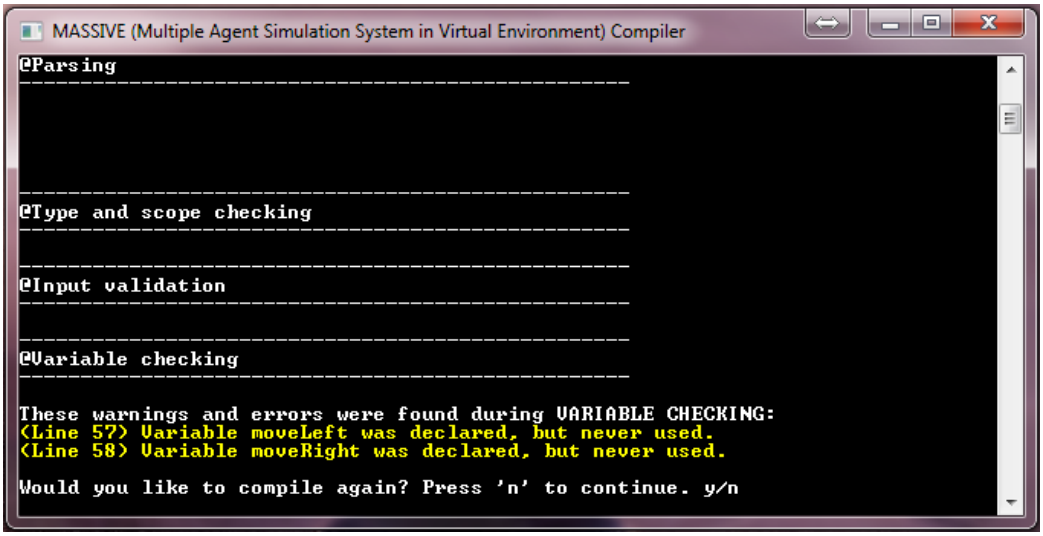


Figure 2.1: The MASSIVE compiler warning of unused variables.

The compiler will happily compile the code again if that option is selected, which provides the programmer with an easy way of correcting erroneous code. After a succesfull compilation a file named "MASSIVECode.cs"and "MASSIVECode.exe" will have been created. The only purpose of creating the cs-file is allowing the programmer to have a look at the code our compiler generates. The cs-file will have been compiled into the exe-file wich is run

automatically. This exe-file creates the actual data output in XML format, which is then run by the MASSIVE simulator, and the user of the simulator is given a choice of how large the game grid will be (see **??**).

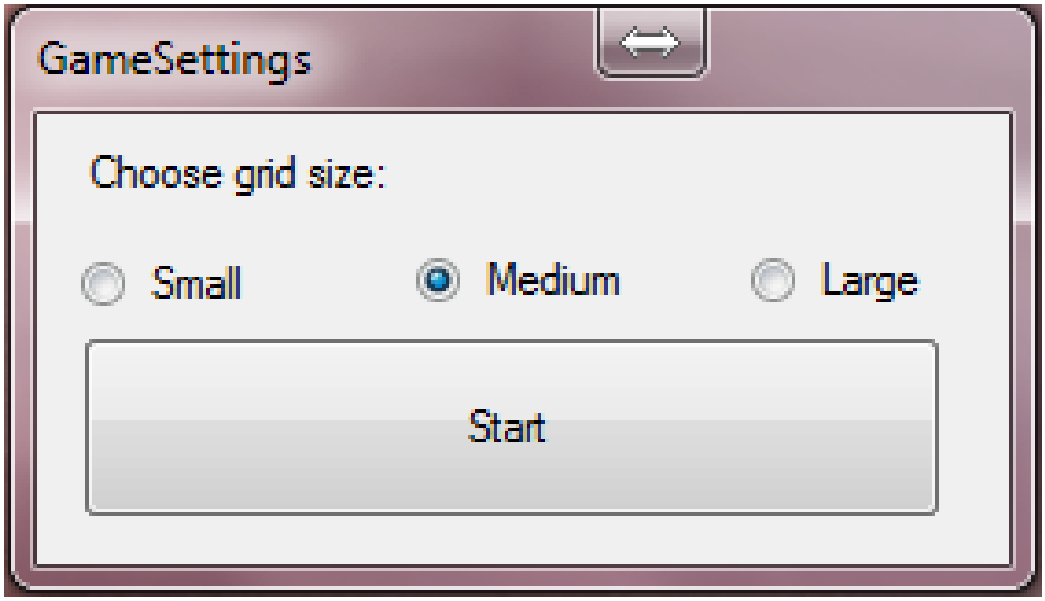


Figure 2.2: Choosing the size of the game-grid

Upon choosing "large", the user will bee presented with the actual simulation (see **??**). Here he will have the oppertunity to instruct the agents to use the action pattern defined in the previous code example, as shown in **??**.

At this point the user is presented with a choice; He can either press "Simulate" to let the simulation run to an end without any interaction, or he can choose to run the game turn-by-turn and control the agents as the game progresses. We see the result of this simulation in **??**.

2.2 Comparison

This section is about how to build a multi agent wargame using C# compared to our own language MASSIVE. We will take a look on some of the pros and cons by using C# aswell as the pros and cons using MASSIVE. We will then compare C# and MASSIVE to examine which language is the best to build a multi agent wargame.

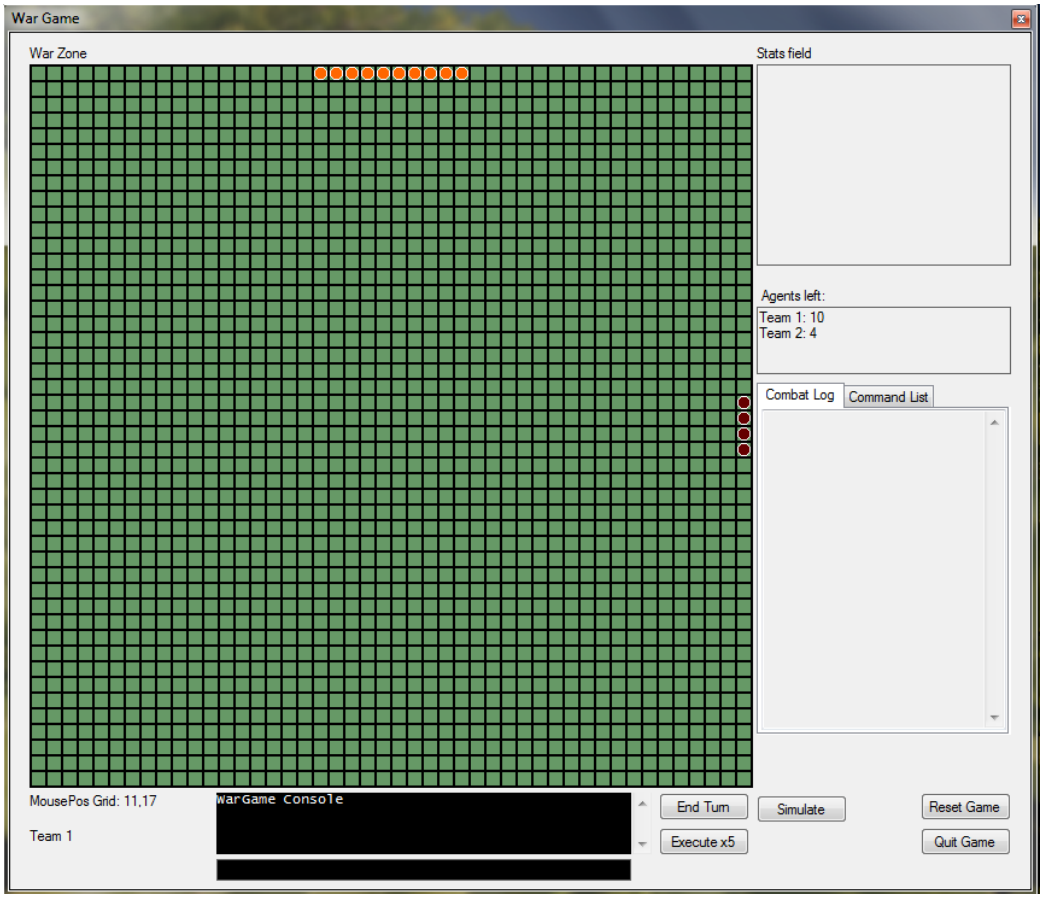


Figure 2.3: The simulation running with the input instructing som of the agents to use an actionpattern

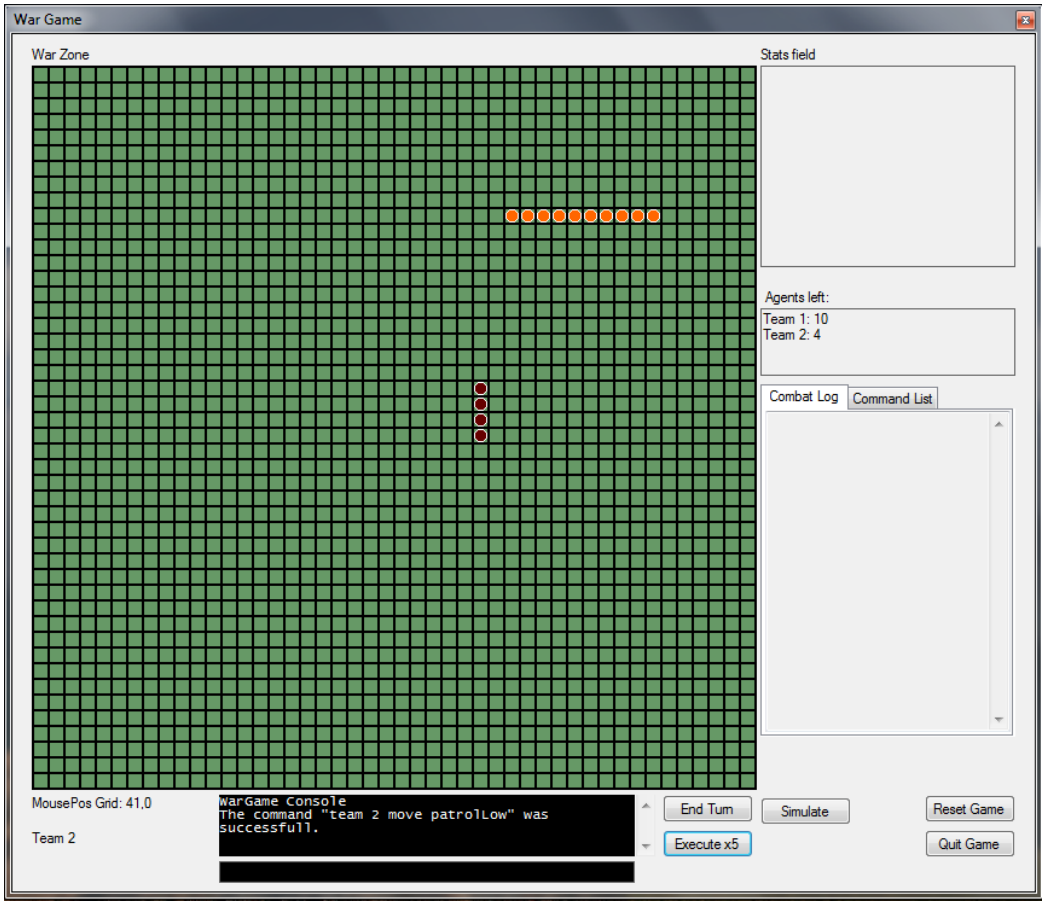


Figure 2.4: The result of the use case simulation in MASSIVE

2.3 C#

We have decided to compare MASSIVE with C# which is an object orientated language(OOP). We decided to use C# to compare with because both our compile and enviroment are written in this language. C# do not have built in multi agent orientated functions or enviroments, which means it is required for the programmer to build the multi agent wargame from scratch. To build a basic multi agent wargame in C# you need to make constructors for agent and teams, furthermore you will also need to create functions for agents and teams, which could be movement and attack functions. At last you would need to create an enviroment to simulate a wargame. However building a multi agent wargame in C# enables you to create all the features you want in a wargame simulation.

Pros

- No limits, you can create all the features you want.

Cons

- No existing multi agent enviroment.
- No existing multi agent types.
- No existing multi agent functions.

2.4 MASSIVE

MASSIVE is a agent orientated language(AOL) which contains premade enviroment and functions for creating agents, squads, teams, and actionpatterns, which means that you do not have to build these yourself and it is therefore relative fast to simulate a wargame. You cannot declare new functions in MASSIVE which limit you to only use the built in functions. The types and functions of MASSIVE are not case sensitve, so you do not need to worry about writing in upper or lower case.

Pros

- Relative fast to simulate a wargame.
- Premade enviroment.

- Premade types for agent, squad, team, actionpattern.
- Types and functions are not case sensitive.

Cons

- Limited to the languages functions.

2.5 C# vs MASSIVE

We will in this section compare a C# code example to the MASSIVE code example earlier in this chapter. We assume that we have already created constuctors, functions, and an enviroment for the C# code.

```
1
2  /* Initializes the game/
3  static void Main(string[] args)
4  {
5
6    // Creates team Disco.
7    Team teamDisco = new Team("Disco", "#FF6600");
8    int totalDiscos = 10;
9    for ( int i = 0; i < totalDiscos; i++)
10   {
11     int a = 0;
12     if ( i < totalDiscos-1 )
13     {
14       a = 1;
15     }
16     else
17     {
18       a = 21-totalDiscos;
19     }
20
21     Agent newAgent = new Agent("Stue", a);
22     teamDisco.add(newAgent);
23   }
24
25   Team teamKman = new Team("Kman", "#660000");
26   Squad squadNabs = new Squad("noobs");
27   Squad squadRevo = new Squad("Revolution");
28
29   for(int i = 0; i < 4; i = i + 1)
30   {
31     int a = 0;
32     if(i =< 1)
33     {
34       a = 2;
```

```
35        }
36        if(i >= 2)
37        {
38          a = 8;
39        }
40
41        Agent newAgent = new Agent("Kman", a);
42        teamKman.add(newAgent);
43
44        if (i <= 1)
45        {
46          squadNabs.add(newAgent);
47        }
48        if (i => 2)
49        {
50          squadRevo.add(newAgent);
51        }
52    }
53
54    // Moves used in the actionPatterns.
55    string moveUp = "unit move up";
56    string moveDown = "unit move down";
57    string moveLeft = "unit move left";
58    string moveRight = "unit move right";
59
60    // Creates the action pattern Patrol Low.
61    // Patrols the lower part of the game area.
62    ActionPattern patrolLow = new ActionPattern("PatrolLow");
63    patrolLow.add(moveUp);
64    patrolLow.add("unit move 25,24");
65    patrolLow.add(moveUp);
66    patrolLow.add("unit move 0,23");
67    patrolLow.add(moveDown);
68 }
```

Source code 2.2: C# code example

In the above code examples you can see how one could generate teams, agents, squads, and actionspatterns using C#. The structure of C# and MASSIVE are very much alike, the only visible differences are how to declare objects, use num instead of int, types and functions are not case sensitive, and you cannot increment a num by using "++". The importan difference cannot be seen in the code example above, because the code example only show how you call functions, declare objects, and perform loops. The importan difference between C# and MASSIVE is that you do not have to create your own environment, types and functions like you do with C#, which would take a long time compared to MASSIVE, you can therefore simulate wargames

relative fast.

CHAPTER 3

Conclusion

In this project a language called MASSIVE is developed. The purpose of MASSIVE is to control agents in a multi agent wargame. In order to implement this language, a compiler is also developed.

The language is limited to creating agents, teams, squads, and actionpatterns for a wargame, because the purpose is to optimize the process of programming multi agent wargame scenarios. MASSIVE is easier to start using than for instance C#, since MASSIVE does not have the same amount of features, and is therefore easier to get an overview of.

MASSIVE comes with constructs for both agents, teams, squads and actionpatterns, allowing for new instances of these to easily be created. MASSIVE also comes with a few methods for easier manipulation of the data, making for more concise code, because the user does not have to define any custom constructs.

A second language has also been developed, designed only to control the agents in real time when running the wargame, which is implemented via an interpreter.

It is evident that MASSIVE is more optimized for programming multi agent wargame scenarios than C#. This is seen from the amount of code needed to prepare a wargame scenario in either language, as seen in section **??**.

tail...