Chapter 1

The WAR programming language

1.1 The Idea

The main idea and thereby the purpose of this programming language-to-be was to create a scripting language for some sort of simulation. During our process of consideration we thought out many different concepts, and ultimately we came to a decision that it would be more interesting to have a controlled environment with a lot of units participating. Reasons for this is that the simulation would not be very interesting to perform when dealing with minor numbers, the reasoning behind this is that we do not expect to make a very complicated simulation, therefore dealing in larger numbers proved more interesting to us.

Ultimately we decided to do a war simulation, with two or more opposing forces spawning in a grid like fashion. These forces would be composed of regiments of different types, with each regiment having a captain. The reasoning for having a captain is to introduce an element which could lead to some interesting even handling, eg. if the captain dies, something happens to the regiment, such as a decline of morale, speed, or behaviour.

1.2 Description of the WAR# language

This language should we read by a compiler which should simulate the script.

In a config file made in common from the two participant they can choose how large or how small they want the grid size of the match, and how many unit they both may own, and when the game should end. This config file should be the rules of the game, so you can change rules from game to game. Each participant shall make a script file they will give to the simulator, which then will compile the script and simulate the battle. These script files contain how many units and what kind of units the participant would like to use.

The one who made the best script will win the game, so every participant have to script the best script, and place the units in the best place on the grid.

How we plan to implement the idea

In order to implement the idea described in section **??**, we plan to make use of an interpreter. Our script will be written in our WAR-code, then compiled to CIL-code, which again is compiled to CLR, which can be run on a machine, capable of running .NET-programs.

1.3 Code Examples

In this section we will show how the scripts written in the WAR scripting language will look like. Normally this section will only contain one code example or more code examples of the same type, but because we are quite uncertain how the script could like we have included 3 types of scripts.

1.3.1 Code example 1

1.3.2 Elements

First code example shows two regiments and the regiments interacts. The code starts with a *#begin* and ends with a *#end*.

1.3.3 Code

```
1 #begin
2
3 // The map is a 50x50 grid
```

```
4
5  //Instantiates  two regiments one of 5 Archer units  and one of 7 Pikemen units:
6  AddRegiment(5, Archer);
7  AddRegiment(7, Pikemen);
8
9  //Places the archer  regiment from coordinate  1,1  to  5,1
10 //Places the pikemen regiment from coordinate  1,3  to  7,3
11 Placement[(1,1),(5,1),(  Archer)][(1,3),(7,3),(    Pikemen)]
12
13 //Checks in the archer  enemy detects some units
14 if  ((Regiment(Archer) spotsUnits())
15 {
16         //Attack the units  it  spotted
17         AttackSpottedUnits();
18
19         //If  the regiment it  self  gets attacked it  retreats
20         if ((Regiment(Archer) getsAttacked())
21         {
22                 Retreat();
23         }
24 }
25
26 //Checks in the archer  enemy detects some units
27 if  ((Regiment(Pikemen) !spotsUnits())
28 {
29         //Moves the regiment forward
30         moveForward();
31         if ((Regiment(Pikemen)) getsAttacked())
32         {
33                 GettingAttackedCounterAttack();
34         }
35 }
36
37
38 //If  either  regiment reaches  the end of  the grid  turn  around
39 if  (Regiment(Archer),(Pikemen)) reachEnd();
40 {
41         turnArround();
42 }
43
```

```
#end
```

Listing 1.1: Code example 1

1.3.4 Code example 2

1.3.5 Elements

In this example we have two different files. The config file specifies the rules of the game such as grid size or allowed sizes of regiments. In the team tile a team is specified. In this file a regiment is defined and a hero unit. Inside the regiment file is the stats of the regiment and a behaviour block exists, inside the behaviour of the regiment is defined.

1.3.6 Code

Team file

```
Team "Ninja Monkeys"

Regiment "Silent  Monkeys"
//Archers
{

        Size  = 200;
        Type = Ranged;
        Range = 120;
        Damage = 2;
        Movement = 30;
        AttackSpeed = 1;

        //Defining the behaviour of the regiment
        Behaviour
        {
                (Unit/regiment/whatever) enemy = Search_for_Enemies();
                if  (enemy.Position <= Range && enemy.Type == Melee)
                {
                        Attack.Position (enemy.Position);
                }
```

```
22                  else  if  (enemy.Position  <= Range+Movement)
23                  {
24                          while(enemyPosition  < Range)
25                          {
26                                  MoveTowards(enemy.Position)
27                          }
28                          Attack.Position (enemy.Position);
29                  }
30                  if  (enemy.position  <= Range && enemy.Type == Hero)
31                  {
32                          MoveAway(enemy.position)
33                          if (enemy.position  <= Range)
34                          {
35                                  Attack.Enemy(enemy);
36                          }
37                  }
38          }
39  }
40
41  Hero "Old Neon Monkey"
42  //Hero unit
43  {
44          Type=Hero.Melee;
45          Movement = 50;
46          Damage = 30
47          AttackSpeed = 5;
48          Behaviour
49          {
50                  Regiment enemy = Search_for_Enemy();
51                  if (enemy.Position == MeleeRange)
52                  {
53                          Attack.Enemy(enemy);
54                          Rape(enemy);
55                  }
56                  else
57                  {
58                          MoveTowards(enemy.Position);
59                  }
60          }
61  }
```

Listing 1.2: Team file of code example 3

Config file

```
Config

//Definition of the grid
Grid "Banana Island"
{
        HorizontalSize  = 512;
        VerticalSize    = 128;
        ObstaclesDensity = 2; //Goes from 1-10
}

Rules
{
        //Standards defines the standard fields for regiments
        //If a regiment haven't defined for example Type it will use the Type in Standards
        Standards
        {
                Size = 200;
                Type = Melee;
                Behaviour
                {
                        //Standard behaviour
                }
        }
        //Rules which the regiments can't exceed
        Maximums
        {
                RegimentSize = 500
                Heros = 1;
                Regiments = 4;
        }
}
```

Listing 1.3: Config file of the code example 3

1.3.7 Code example 3

1.3.8 Elements

This code is resembles the code in code example 2. The biggest difference is that elements found in most normal programming languages are left out, like ; or datatypes (int,string).

1.3.9 Code

```
unit pirate = 10
unit soldier = 5

Pirate
{
        Speed = 4
        AttackPower = 1
        DefensivePower = 2

        if (Enemy is near)
        {
                MoveTowards
        }
        else
        {
                wait
        }
}

Soldier
{
        Speed = 1
        AttackPower = 10
        DefensivePower = 5
        if (enemy is inRange)
        {
                Attack
        }
        else if (Enemy is near)
        {
```

```
31                  moveTowards
32              }
33  }
```

Listing 1.4: Code example 3

Chapter 2

Specification of the WAR language

2.1 Syntax

Terminals

if	{	}	Behaviour
else	=	¿	Standards
/	*	-	Maximums
(	)	;	¡ =
+=	-=	*=	¿ =
/=	,	.	Regiment
Team	Rules	Config	Grid
¿	+		

Nonterminals

Program	Command
V-name	Declaration
Operator	Single-Command
Integer-Literal	Single-Declaration
Expression	Primary-Expression
Comment	