COMP-361 Software Engineering Project
Medieval Warfare v1.0

Medieval Warfare is a turn-based, multi-player, resource gathering strategy game. In the Middle Ages a war is at hand. A conflict has arisen and the peasants are at arms and taking sides. It is your job to unify the front and ensure victory prevails to your kingdoms.

Players start with a preset amount of land on which to build their empire. Each region the player controls contains one village. From their villages a player can hire villagers to join his cause. A villager can then be moved to take over a neutral land or gather wood in the forest. Soldiers can be recruited to invade land. The game ends when one player has taken over the entire country, i.e., all other players were eliminated from the game.

Map

The game is played on a hexagonal playing field that is in the shape of an island comprised of at least 300 hexagons. The island can be of any shape, provided that all land tiles are connected. The island is surrounded by sea tiles, which are not counted as part of the playable area in the map. The land tiles are initially randomly colored with "number of players + 1" colors, where one color represents neutral land and the other colors represent land owned by a player. Some tiles will contain trees (20% randomly placed) or meadows (10% randomly placed), and any group of same-colored player tiles that touch and form a region comprised of at least 3 tiles should contain a village on a random tile in the region. These initially placed villages contain 7 gold at the start of the game.

To finalize the map initialization, any player-colored region not large enough to support a village should always become neutral territory. An example of parts of a 2-player game map are shown in figure 1.

Villages

A village is placed on a region of similarly colored tiles that form a group of 3 or more. It is placed randomly on a tile replacing whatever is there whenever a region of at least 3 tiles of the same color is formed (or if the village of a region is destroyed because of an enemy invasion, and the remaining region no longer has a village).

A village has a treasury of gold and a wood pile, which are initially empty (except the initial villages, which contain 7 gold at the start of the game). At the start of each turn of a player, the village generates an income of gold equal to the number of tiles in the region it occupies. Any tile which contains a meadow is worth 2 gold instead of 1. A tile with a tree doesn't generate any gold. The gold generated is added to the treasury of the village and accumulates over time.

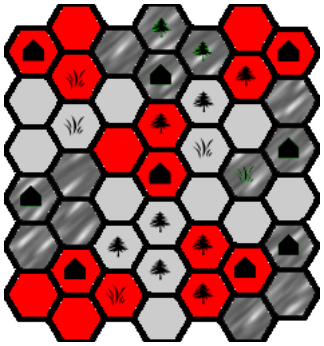

Figure 1: Parts of a 2-Player Game Map

Also, at the start of each turn, after income is generated, the village must pay the wage of all its villagers (i.e., that are located on its region). If it does not have enough money in its treasury, all the villagers supported by that village perish and the village is left with no money.

Upgrading Villages

The player has the option to spend wood to upgrade his village to the next level; however, nothing further can be done with that village on this turn or the next turn once it has been upgraded (i.e. recruiting villagers, building structures). The following upgrades are available:

- **Hovel**: no cost
  The initial structure all regions have. They can recruit/train peasants and infantry and can be overtaken by enemy soldiers.
- **Town**: Cost 8 Wood
  A Town can recruit/train Soldiers and can build towers (see below).
- **Fort**: Built Town and 8 Wood
  The Fort is a strong structure which can only be sieged by an enemy Knight as it commands an area around it of one hex like a Soldier. It also allows you to recruit/train Knights.

Villagers

From a village the player can spend gold to hire/train a villager to join his army. Initially, only peasants and infantry are available. A hired villager is placed on any free tile in the same region as the village he was purchased from or an adjacent tile if it is possible to invade.

Villagers can be trained to a higher skill level / rank by spending more gold. There are four types of villagers:

- **Peasant**: Cost 10 gold, upkeep 2 gold per turn
  The weak underlings of your army, armed with pitchforks. They're willing to do pretty much anything (gather wood, clear tombstones, cultivate meadow) except invading enemy territory.
- **Infantry**: Cost 20 gold (Cost 10 to upgrade from peasant), upkeep 6 gold per turn
  The first attack force, good for light military operations, but can't takeover a village. An infantry is still willing to gather wood and clear tombstones, but won't cultivate meadows.
- **Soldier**: Cost 30 gold (Cost 10 to upgrade from infantry, 20 from peasant), upkeep 18 gold per turn
  The strongest man still willing to do hard labor (gather wood and clear tombstones). Soldiers can also invade villages. Unfortunately, soldiers trample any meadow they come in contact with, unless there is a road leading through the meadow.
- **Knight**: Cost 40 gold (Cost 10 to upgrade from soldier, 20 from infantry, 30 from peasant), upkeep 54 gold per turn
  A knight refuses to do menial takes, he's purely for war, so don't count on him to gather wood or even venture into the forest. A knight also flattens any meadow they come in contact with, unless there is a road leading through the meadow.

Combining / Upgrading Villagers

It is possible to take your existing villagers and combine them to form a stronger unit. It is not possible to separate them again so be careful. Two peasants can combine to form an infantry, a peasant and an infantry to form a soldier, a peasant and a soldier to form a knight, or two infantries to form a knight. It is, of course, not possible to combine two villagers to result in a villager with a higher rank than the region's village can support.

In a similar manner, it is also possible to select a villager on a land and spend the difference in gold to upgrade/train him to a higher rank as high as the village of his region supports.

Moving Villagers

When placed on a tile, a villager commands the hex he is on and all adjacent ones of the same color. A tile can hold only one villager, structure, or tree at a time. Meadows and roads are an exception, since a villager can walk over meadows (except a knight, which destroys a meadow if he rides over it) and roads.

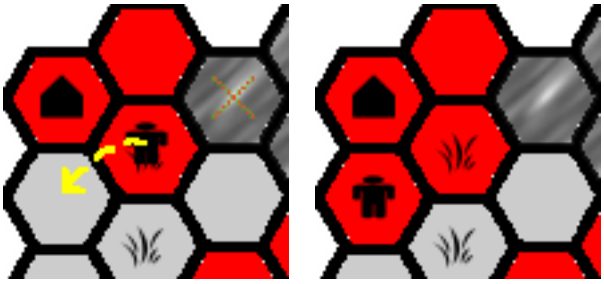

Figure 2: A Peasant Taking Over Neutral Land

He can continue to move on the same turn as many times as the player wishes[1] unless he gathers wood, cultivates a meadow, builds a road, clears a tombstone, or takes over new land.

These activities are described below:

- **Gathering Wood**: If a villager is placed on a tree square (except a Knight), the square is vacated (as it is replaced with the villager) and 1 wood is added to the region's village's stockpile.
- **Clearing a Tombstone**: When a villager perishes because the supporting village runs out of gold, he is replaced by a tombstone. If a villager walks over a tombstone (except a knight), the tombstone is removed (and the villager can no longer move). Any tombstone that still exists at the beginning of the following turn of the player transforms into a tree (after the tree growth phase).
- **Cultivate Meadow**: A peasant can spend two turns cultivating a meadow. If instructed to do so, the peasant will be unable to move for the remainder of the turn and the next. At the start of the third turn, the peasant is freed, and a meadow is placed on the tile underneath the peasant (and thus the extra gold from the meadow is counted that third turn).
- **Building a Road**: A peasant can spend one turn building a road. Roads can be built on empty tiles (to prevent forests from growing, for example) or on meadows (to prevent knights from trampling them). If instructed to do so, the peasant will be unable to move for the remainder of the turn. At the start of the next turn, the peasant is freed, and a road is placed on the tile underneath the peasant.

Invading

The only land a peasant can invade is a neutral territory adjacent to his region. Upon doing so that peasant will no longer be able to move for the remainder of the turn and the tile is converted to the color of the player as illustrated in figure 2.

If there is a tree on that land, one wood is collected as well. If the acquisition of the new land joins two separate regions of the same player, the resources of the villages are pooled together in the village that was the most advanced (or, if both are of the same kind, the one that controlled the biggest region before the move); the other village is replaced by a meadow. This situation is illustrated in figure 3.

Villagers of the rank of infantry or higher can invade enemy territory. An infantry can only invade a new territory if that land is not protected by an enemy villager (in 1 hex distance) of equal or greater skill, i.e., an infantry can't invade next to an enemy infantry, but an infantry can invade a land guarded by a peasant, or a soldier can invade a land protected by an infantry. If the invaded land contains an inferior enemy villager, that villager is destroyed (as he is being replaced by the invading villager). In either case, as soon as the land is successfully invaded, it is owned by the invading player and the tile color should change accordingly. If the enemy region can no longer support a village after being invaded (less than 3 hexes), the village is turned into a tree (all resources in that village are thus lost) and the region is converted to neutral territory. This situation is illustrated in figure 4.

If a player does not own any regions anymore, he is eliminated from the game.

Invading Villages

A village can be invaded by a soldier or knight assuming the tile is not under the protection of an equal or stronger unit. When this occurs, all the money and resources of that village are taken and placed in the treasury of the invading soldier's

[1] In other words, a villager can move to any tile within his region provided there is an obstacle-free path from leading from his current location to that tile.

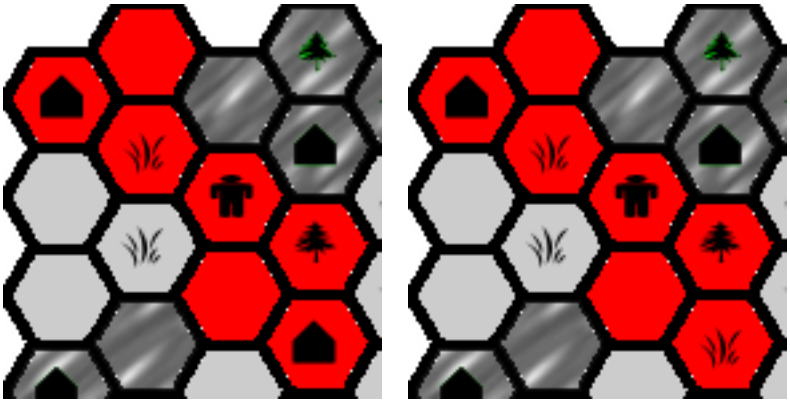

Figure 3: Joining of Two Regions after Peasent Acquired New Tile

Figure 4: Green Infantry Invades Red Land – Remaining Land Cannot Support Village Anymore

village, the tile is converted to a meadow of the invading player's color, and the enemy village is recreated (as an empty hovel) at a random position on the remaining landmass of the region, provided that the region is still large enough, i.e., at least 3 hexes big.

Structures

A town or fort can build watchtowers at the cost of 5 wood on any tile of the region it controls. A tower commands an area around it of 1 hex (just like a villager) and acts as the same level as an infantry, but with no upkeep. Once built, towers cannot be moved or upgraded. Also, like a regular infantry unit they can be destroyed by a higher ranking enemy, i.e., by a soldier or knight.

Turn Overview

Medieval Warfare is a turn-based game. At the beginning of the game, an ordering is established between the players. During the entire game, players take turns in that order. Once each player has had one turn, a "round" has been completed.

A round consists of several phases as described below.

1. Tree Growth Phase: This phase occurs at the beginning of each round (except for the very first round). For each tree tile, an adjacent empty or meadow tile is picked randomly and a new tree is created on that tile with a 50% probability. If there are no adjacent empty or meadow tiles (i.e. all adjacent tiles are either sea tiles, other trees, villages, structures, roads, or occupied by villagers) then no additional tree is created on this turn for that tree.

2. Player Phase: For each player, in the order established at the beginning of the game, the following steps are executed:

   (a) Tombstone Phase: Any tombstones on tiles owned by the player are replaced by trees.

   (b) Build Phase: Meadows are produced if a peasant belonging to the player has now spent 2 turns cultivating. Roads are produced if a peasant belong to the player has now spent 1 turn building it.

   (c) Income Phase: Money is added to each village of the player based on the number of tiles in the region (+2 for each meadow, +1 for each empty tile, 0 for each tree).

(d) Payment Phase: Money is subtracted from each village's treasury based on the villagers that it supports. If a village has insufficient funds to pay the villagers it supports, *all* villagers supported by that village perish and are replaced by tombstones.

(e) Move & Purchase Phase: This is the playable phase of the turn. The player can now recruit/train new villagers, combine/upgrade villagers, and move them around their region until an action has been performed for each unit (taking new land, cultivating meadow, chopping wood, building a road, clearing a tombstone, simply remaining in place, etc.).

Additional Rule Clarifications

Additional rule clarifications, if any, are going to be posted on the course webpage during the year at: http://www.cs.mcgill.ca/~joerg/SEL/COMP-361_FAQ.html.

Application to Develop

The rules above describe the game play, but they do not prescribe computer-specific functionality, such as the user interface. In your implementation you have to design your own user interface. Make it as intuitive as possible; provide guidance for the player; provide context-sensitive help; voice messaging; speech recognition, 5.1 channel sound, "the sky is the limit", ... You are also allowed to extend the game by coming up with additional game features, e.g., new units, other structures, new land types, trading, new winning conditions, etc... In case you want to do that, we require you to document the game rule changes / extensions precisely, discuss them with us (we will ensure that the changes you suggest do not simplify the game design and implementation), and commit to them before the end of the first semester.

Adding new rules to the game is fun, however, 80% of your grade is solely based on whether or not you implemented the game rules exactly as specified. Additionally, certain computer-specific functionality is required as discussed in the following sections.

1. User Interface Assistance

The software must help a player to control his villages/regions and villagers in an efficient way. This includes, for example, the possibility of viewing a list of all villages, checking their respective treasuries, to easily determine if there are villages that are having trouble supporting their villagers.

2. Distribution and Game Server

The application must make it possible for (at least) three players to play against each other over a network. This means that somehow, the three players must be able to set up a connection between three computers (or game consoles), agree on a game map, and then play their moves turn by turn until one player wins. Alliances (e.g. 2 against 2), are optional.

To facilitate players to setup a game, you should implement a game server that allows players to connect with other players that are currently online. The server keeps score of how many games each player has played, and how many games she has won in total. After logging in, a player should see the list of players that are currently online together with their game statistics.

Somehow, two (or more) players that want to play a game together should be able to agree on a game map, and then play their moves turn by turn until one player wins. When a game is completed, the server should update the players statistics.

3. Different Islands

The players should be able to choose among several islands (or generate random ones) when starting a new game. Islands must be of sufficient size (at least 300 hexagons), but do not need to be of a specific shape as long as it is possible to reach every square.

4. Saving

The application must allow any player to save a game. Later, it should be possible to reload a previously saved game, and continue playing from there.

Project Milestones

The project is to be done in groups composed of maximum 5 students. All members of a group will get the same final grade. This means of course that all group members are supposed to contribute equally to the success of the project. Usually this works best if all group members have similar ambitions for the project. We recommend therefore that you discuss this openly within the group before committing to be part of the team. During the year, if you are experiencing problems within your group (e.g., communication problems, or you are feeling that the project workload is not equally shared among team members), then don't wait and talk to me or one of the TAs about it. If we know about a potential problem then we can help to solve it!

The final grade of the project is composed of:

- 3% for the user interface sketch (mid October)
- 15% for the requirements document (late November)
- 12% for the design document (early January)
- 15% for the demo (March)
- 20% for the acceptance test (April)

Details for each required hand-in are going to be announced separately, but below you'll find a brief summary of each milestone so you can plan accordingly.

As announced in class, you can use any implementation platform you like, provided that the programming language is object-oriented. However, we will provide technical support for students using Java. Also, we developed a Java-based game programming framework called Minueto (http://minueto.cs.mcgill.ca/), that takes care of efficient rendering of graphics (off-screen drawing, transparency, hardware acceleration, etc....). That way, students who do not know much about Java graphics will not have to waste time learning these concepts, but can focus on the more important functional part of the application. Minueto also does some basic user input handling and provides sound output.

Finally, please also keep in mind that the demonstrations and the acceptance test have to take place either in the Trottier building, or somewhere else on the McGill campus (i.e., you can, for instance, bring your laptop / desktop computers to my office and set up the demo there, or run the game on your PDA's, bring your gameboys / XBoxes / iPhones to school, etc....).

1. User Interface Sketch

Good user interfaces should prohibit the user from making "mistakes", such as unauthorized or useless moves. An intuitive user interface should also assist players whenever possible, for instance by highlighting possible destination tiles for a selected villager. Also, since the number of villages and villagers of a player can become fairly large, a good user interface should allow a player to manage his resources and units in an efficient way.

In mid October you should hand in a sketch (either hand drawn or a printout) of the main screens that are presented to the players of the game. It should show how the player interacts with the game server, how she sees the island and regions when playing the game, how she can control villages and villagers, and how she accesses all other required game functions (e.g., saving the game).

2. Requirements Document

This hand-in comprises requirements models that clearly specify the functionalities that your game implementation needs to provide. This includes a use case model, a concept model, an environment model, a protocol model and an operation model.

3. Design Document

This hand-in comprises design models that provide a detailed blueprint of the structure (i.e. classes) and behaviour (i.e. methods) of your application that relate to the implementation of the game and its rules. (The design models will not cover classes and methods that are needed to handle graphics and network communication).

4. Demo

After spring break you will be asked to give a first demonstration of your product. You will be given a list of minimal functionality that you must show during the demo. 80% of the grade for the demo is based on the correct implementation of the required functionality, and that you can run the demonstration without visible errors / bugs / crashes. The remaining 20% are attributed as follows: 5% for the quality of the demo (i.e. presentation / explanation of what is happening), and 15% for additional functionality of the game that you can demonstrate, but that was not required for the demo.

During the demo, the group of graders will not touch the computers, nor interfere with the demo in any way. YOU run the show. Demonstrate what your software can do (and do not show what it can not do). Explain what is happening during the demo, i.e. how you set-up the communication between the players, what functionality you are demoing, etc. Show off your cool features, and do not talk about the bad ones / remaining bugs. Any unforeseen event that hurts the "demo effect" might affect your grade.

5. Maintenance Phase

In the week of the demo, I will hand out some slight changes / additions to the game rules. This simulates "real-life" software development, in which the application requirements are often subject to change during the development of an application. In order to prepare for this phase, try to come up with a flexible design / write structured, modular, extensible code.

6. Acceptance Test

In the last week of classes your application will have to pass the "acceptance test". During the test, the group of graders will play your game, looking for bugs / glitches and violations of the game rules. We run the show, and you basically just watch.

80% of the grade for the acceptance tests are based on the correct implementation of the game rules (i.e. if your application fulfills all requirements listed in this document, and if there are no errors / bugs / crashes during the acceptance test, you will get at least an A- (80)). The remaining 20% are attributed for an elegant / intuitive user interface, or other extra features. For instance, you could get extra points by having a 3D user interface, or by allowing alliances among players, or by allowing a neutral "observer" to connect to a running game, or by adding additional structures, units or extensions, or by allowing players to trade resources, etc.

In order to prepare for the acceptance test, please keep in mind that it takes a long time to "debug" an application of this size. Even under the assumption that you do extensive unit testing during development, I suggest to plan at least 2 weeks for final adjustments.

Some Suggestions

Here's a list of some simple suggestions you should keep in mind during the development:

- Keep everyone in the group in "a good mood".
- Assign responsibilities to group members.
- Have regular group meetings to consolidate your work.
- Start implementation early, i.e. January at the latest.
- First strive for a simple, correct implementation. Later (if there is enough time), add more sophistication.
- Always keep the deadlines in mind. For the demo you must have a functional, convincing application that provides the required minimal functionality.
- Do not make big changes on the day that precedes the demo or the acceptance test (or else be sure to have a functional backup copy...)
- Testing takes time.
- Plan for the unpredictable!
- Start implementation early :)

Change Log

- v1.0: Initial release.