

CS351 Spring 2005, Project 3

JCiv

MondoSoft, Inc.

April 4, 2005

STATUS Specification and requirements document

VERSION 1.1

DATE Apr 4, 2005

REPLACES This document obsoletes and replaces Version 1.0.

0 Changelog

Version 1.0 Initial release. Apr 4, 2005.

Version 1.1 Bug fixes and critical, last-second updates.

- Completed Definitions (Section 3)
- Fixed a couple of bugs in the Timeline (Section 9)
- Extensive clarification and updates to the Deliverables (Section 7)
- Updates to User Interface specification (Section 6.1).

1 Summary

Deciding to capitalize on their recent successes in the computer gaming world¹, MondoSoft has decided to branch out into a more modern gaming platform. In keeping with the corporate strategy of attempting to dominate the world, you will be developing a game engine and platform for the JSimCivQuestCraft (more concisely known as JCiv) game: a realtime, multi-player, online game of strategy and conquest.

Because this game is quite complex, a critical component of this project is building a good graphical user interface (GUI) to visualize and control the game. The project will be assessed, in part, on the quality and intuitiveness of this UI.

¹They sold at least four copies of Zurk.

Contents

0	Changelog	1
1	Summary	1
2	Group/Individual Effort	5
2.1	Individual Participation Evaluations	5
3	Definitions	5
4	Rules	7
4.1	Time and Event Scheduling	7
4.1.1	The Global Clock	7
4.1.2	Wall Clock Time	8
4.2	Game Board, Terrain, and the Map	8
4.2.1	Terrain Types	8
4.2.2	Visibility	9
4.2.3	Other CELL Contents	9
4.3	Resources	10
4.3.1	Resource Units and the Bank	10
4.3.2	Resource Types	10
4.3.3	Extracting Resources	10
4.3.4	Spending Resources	10
4.3.5	Capturing Resources	11
4.4	Units and Commands	11
4.4.1	Unit Types and Properties	11
4.4.2	Unit Commands	13
4.4.3	Movement and Path Planning	14
4.5	Buildings	14
4.5.1	Building Types and Properties	14
4.5.2	Building Commands	15
4.6	Combat	16
4.6.1	UNIT-to-UNIT	16
4.6.2	UNIT-to-BUILDING	16
4.6.3	Invading	16
4.6.4	BUILDING-to-BUILDING	16
4.7	Victory Condition	16
4.8	Game Parameters	17
5	Network JCiv	17
6	Requirements	17
6.1	User Interface	17
6.2	Multithreading	18

7 Deliverables	19
7.1 Milestone 1: Development Plan	19
7.1.1 Milestone Blackout Dates	19
7.2 Rollout	19
8 Grading	21
9 Timeline	22

List of Tables

1	Minimal set of TERRAIN TYPEs for Jciv, and their respective properties.	9
2	Unit types.	12
3	Commands available to UNITS.	13
4	Building types.	14
5	Commands available to BUILDINGs.	15
6	Summary of game parameters.	17

2 Group/Individual Effort

This is a *group* design and development project. You are required to work closely with your group members on all phases of the project including initial design, timeline development, testing, code development, documentation, etc. The whole project is “owned” by the entire group, and, by default, all group members will receive the same grade for the project. To the extent possible, all group members should be present for intermediate progress reports and milestone deliveries.

2.1 Individual Participation Evaluations

As part of the final project rollout, each *individual* is to turn in an evaluation of his or her own performance and role in the group, as well as an assessment of other group members’ contributions and performances. Individual grades will be adjusted from the group total grade to account for individual contributions. E.g., if an individual is uniformly seen to have not participated in the project, that individual’s project grade will be discounted. On the other hand, if some group member is widely felt to have contributed “above and beyond the call”, that member may receive bonus credit.

This evaluation is to be turned in privately, via direct email to Prof. Lane and Mr. Brown. The content of these evaluations will be held private and not revealed to other group members, except as generalities. That is, the instructors may reveal to a group member that the group was highly dissatisfied (or highly pleased) with his/her performance, but the instructors will not reveal specifically who said what about whom.

3 Definitions

ATTACK VALUE The bonus that a UNIT or BUILDING receives when Attacking another UNIT or BUILDING.

BANK A record of how much of each type of RESOURCE a player has.

BUILDING A fixed structure, occupying a CELL of the game grid. A BUILDING provides some cover to UNITS housed there, as well as being a source of new UNITS and RESOURCES.

CELL The smallest unit of the game board. A single point on the grid. A CELL has a TERRAIN TYPE and may contain a single BUILDING and one or more UNITS.

CLIENTS Prof. Lane and TA Brown.

COMMAND A directive issued to a UNIT. Every COMMAND has an *assigned unit*, a (possibly null) *target*, and a *time to complete*. Additionally, some COMMANDs have RESOURCE costs.

DEFENSE VALUE The bonus that a UNIT or BUILDING receives when it is being attacked by another UNIT or BUILDING. Note that the UNIT/BUILDING does not have to be *actively* defending (i.e., it does not have to be using the Defend COMMAND) – it receives this bonus regardless of what COMMAND it is executing. When the UNIT/BUILDING

is *actively* defending (using the `Defend` COMMAND), it receives *double* its base defense bonus.

DIMENSIONS Size of the game grid. DIMENSIONS comprises two components: s_x (left-to-right) and s_y (top-to-bottom) sizes.

FRU Fundamental Resource Unit. The smallest unit of measure for each resource type.

GAME INITIALIZATION Setup phase during which game-wide parameters such as board DIMENSION and MAP are set.

GLOBAL CLOCK An internal timer, maintained by the game engine, that determines relative ordering of events, how long it takes COMMANDs to run, etc.

GLOBAL TIME The current value of the GLOBAL CLOCK.

MANHATTAN DISTANCE The distance measured when walking through Manhattan (i.e., when the agent has to travel along orthogonal streets rather than along the diagonal). Simply defined by:

$$\begin{aligned}d(\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle) &= \|x_1 - x_2\| + \|y_1 - y_2\| \\ &= \Delta x + \Delta y\end{aligned}$$

, rather than the traditional “Euclidean” distance (given by the Pythagorean theorem). All distances in the `JCiv` game world are calculated using Manhattan distance. A.k.a., city street distance.

MAP The terrain definition for a single game instance. Given the DIMENSIONS of the game grid, the MAP specifies a TERRAIN TYPE for each CELL of that board, as well as starting locations for each PLAYER.

MAY A requirement that the product can choose to implement if desired. Can also indicate a choice among acceptable alternatives (e.g., “The program MAY do x, y, or z.” indicates that the choice of behavior x, y, or z is up to the designer.)

MUST A requirement that the product must implement for full credit.

MUST NOT A behavior or assumption that must not be violated. Violating a MUST NOT restriction will result in a penalty on the assignment.

PLAYER A human player, directing UNITS and BUILDINGs through a single client program.

RECOVERABLE ERROR An error condition that the software can ignore, correct, or otherwise recover from. The program MUST produce a warning message and then cleanly continue with no corruption or loss of valid data.

RESOURCE A material that is necessary to create and maintain UNITS and BUILDINGs. RESOURCES come in a number of types and are obtained either through the `Produce` command applied to various TERRAIN TYPEs or by `Invadeing` a BUILDING and capturing the supplies stored there.

TERRAIN TYPE The type of land at each cell of the game board. TERRAIN TYPE determines properties of the land such as the RESOURCE that it produces, its MOVE COST, etc.

UI User Interface provided by the client to allow a PLAYER to manipulate the game state.

UNIT The smallest group of people in the game. A UNIT is a mobile force that can take actions like Attack, Build, etc.

UNRECOVERABLE ERROR An error condition from which recovery is impossible. The program MUST produce an error message describing the condition and then cleanly halt.

VISIBILITY The part of the MAP that can be seen by a player. Visibility depends on the positions of a player’s UNITS and BUILDINGS.

WALL CLOCK The subjective time experienced by the player, measured by absolute time. The ratio of GLOBAL CLOCK speed to WALL CLOCK speed is determined by the system parameter t_s .

4 Rules

This section discusses the rules of J Civ; later sections describe networking, user interface, and deliverables.

4.1 Time and Event Scheduling

J Civ is an *asynchronous* (or “real-time”) game, meaning that players do not take turns. Instead, each player can dynamically assign COMMANDs to her or his UNITS at arbitrary times. Each UNIT then acts on its COMMAND independently until either the COMMAND is completed or the player gives the UNIT a new COMMAND. In the absence of player intervention, the game state will continue to evolve as UNITS carry out their assigned COMMANDs.

To be precise about the timing and evolution of this asynchronous game, it is necessary to distinguish two types of time:

4.1.1 The Global Clock

Although the game evolves without player intervention, there *is* an internal “clock” that determines when events take place and ensures the correct ordering of events. COMMAND durations, movement speeds, etc. are all defined in terms of ticks of this GLOBAL CLOCK signal. Every time a new event is created, it is added to a global schedule, along with a note specifying when the event will take place. Every time the global clock ticks, the game engine checks to see if any events are scheduled at that time unit and carries out those events.

For example, suppose that the current schedule is as follows:

EVENT	Produce (u_3, c_{135})	Move (u_{17}, c_{12})	Attack (u_{11}, u_9)	Spawn ($b_2, Serf$)
SCHED TIME	1138	1141	1149	1160

and that the current GLOBAL TIME is 1137. On the next clock tick, the GLOBAL TIME becomes 1138 and the event Produce, applied to UNIT u_3 and cell c_{135} , takes place and is removed

from the schedule. After `Produceing`, UNIT u_3 is re-scheduled to `Produce` again at time step 1158 (c_{135} is a forest cell and produces wood; see Section 4.2). Now the schedule appears as follows:

EVENT	<code>Move(u_{17}, c_{12})</code>	<code>Attack(u_{11}, u_9)</code>	<code>Produce(u_3, c_{135})</code>	<code>Spawn($b_2, Serf$)</code>
SCHED TIME	1141	1149	1158	1160

On time steps 1139 and 1140 nothing happens. On time step 1141, the `Move` event takes place and is removed from the schedule. This scheduling and action loop continues until the game ends (Section 4.7).

If an event is encountered whose conditions no longer apply (e.g., a `Produce` event is scheduled for UNIT u_3 , but u_3 no longer exists when the event is encountered), the event **MUST** be silently dropped. However, care must be taken to *deschedule* any events for a UNIT whose `COMMAND` is changed. E.g., suppose that UNIT u_{97} is scheduled to execute a `Build` command at GLOBAL TIME 27893, but at GLOBAL TIME 27695, the player changes u_{97} 's `COMMAND` to `Move`. The `Build` `COMMAND` **MUST** be descheduled before the `Move` command is scheduled to ensure that u_{97} does not execute two mutually exclusive `COMMAND`s.

The `GLOBAL CLOCK` is maintained on the server to ensure that all clients schedule events according to the same time base.

4.1.2 Wall Clock Time

`WALL CLOCK` time is the time experienced by the player and determines the subjective impression of how “fast” the game runs. The relative speed of the game is determined by the game parameter t_s : the “game speed”. t_s is determined to be the number of milliseconds of `WALL CLOCK` time between each tick of the `GLOBAL CLOCK`. t_s is a game parameter set during `INITIALIZATION`.

4.2 Game Board, Terrain, and the Map

`JCiv` is played on toroidal grid. That is, the game board is a regular rectangular grid of `CELL`s, but the top of the grid meets the bottom and the left side meets the right side. A UNIT that moves off the top of the board will find itself on the bottom, etc. The `DIMENSIONS` of the board (number of `CELL`s in each direction before wraparound) are specified by two parameters, s_X and s_Y , that are set during `GAME INITIALIZATION`.

4.2.1 Terrain Types

Each `CELL` is assigned a single, immutable `TERRAIN TYPE` for the duration of a single game. The assignment of `TERRAIN TYPE`s to each `CELL` constitutes a `MAP`. Each game session has a single `MAP`. The `JCiv` game suite **MUST** provide a mechanism to load a `MAP` from disk.

`TERRAIN TYPE` defines how hard it is to move through that `CELL`, what kinds and amounts of `RESOURCES` can be extracted from that `CELL`, how long it takes to produce one `FRU` of that `CELL`'s `RESOURCE`, and how the `CELL` is displayed in the user interface. `RESOURCE` properties (type, limit, and production time) are discussed in Section 4.3; movement cost is discussed in Section 4.4. A `TERRAIN TYPE` may have other properties as well, at the designers' discretion. E.g., some `TERRAIN TYPE`s might change `DEFENSE VALUE` or `VISIBILITY`.

TERRAIN TYPE	Resource Type	Resource Limit	Production Time	Move Cost
Plains	food	3	10	1
Desert	none	—	—	1
Forest	wood	2	20	2
Mountain	iron	1	30	3
Swamp	none	—	—	3

Table 1: Minimal set of TERRAIN TYPEs for $\mathcal{J}Civ$, and their respective properties.

The $\mathcal{J}Civ$ game suite MUST implement *at least* the TERRAIN TYPEs specified in Table 1.

The visual display for the TERRAIN TYPEs is unspecified and may be chosen by the designers. All TERRAIN TYPEs MUST, however, be visually distinct. The TERRAIN TYPEs MUST be represented by a graphic representation rather than (or in addition to) a plain text string. This graphic may be arbitrarily simple (e.g., a plain blob of a pure color, with a different color for each TERRAIN TYPE), but bonus credit MAY be awarded for a nice visual design.

OPTIONAL RULE Additional TERRAIN TYPEs MAY be provided, at the designers' discretion. E.g., the $\mathcal{J}Civ$ game MAY support water (ocean, lake, or river) TERRAIN TYPEs that can only be navigated by special UNITs (such as ships), or TERRAIN TYPEs that produce special types of RESOURCEs (stone, gold, rubber, etc.)

4.2.2 Visibility

While the game engine itself maintains the complete MAP and knows where all UNITs, BUILDINGs, etc. are, the players do not have total knowledge. The players can see only what their UNITs have seen. The UI display for each player MUST hide the contents (including TERRAIN TYPE) of any CELLS that have not been seen by any of their UNITs. E.g., such CELLS may simply appear as black squares to the player.

Once a CELL has been seen, that CELL's TERRAIN TYPE is known to the player and appears on that player's display forever thereafter.

When a UNIT is seen, its position and type is marked so long as it can still be seen by any of the player's UNITs. As soon as no friendly UNIT can see the enemy UNIT, the enemy UNIT is removed from the player's display.

A player can always see all of his/her UNITs and the TERRAIN TYPEs and other contents of the CELLS within viewable radius of any of his/her UNITs.

4.2.3 Other CELL Contents

In addition to TERRAIN TYPE, each CELL may hold 0 or more UNITs (Section 4.4) and 0 or 1 BUILDING (Section 4.5).

Note that, although an arbitrary number of UNITs can occupy a single CELL, only UNITs belonging to a single PLAYER can occupy the same CELL at the same time.

4.3 Resources

JCiv is a *resource management* game. That is, each TERRAIN TYPE produces a single type of RESOURCE, which the players can harvest with the appropriate type(s) of UNITS. In turn, the player must spend RESOURCES to build new UNITS or BUILDINGS.

4.3.1 Resource Units and the Bank

RESOURCES are measured in FRUs: “fundamental resource units”. Each player has a RESOURCE BANK that records how much of each type of RESOURCE a player has at the current moment. E.g., at some time, a player’s BANK might record that she has 3 FRUs of wood, 5 of food, and 0 of iron. Every time a UNIT harvests an FRU of some RESOURCE type, the corresponding bin in that player’s BANK is incremented. Every time the player constructs a new UNIT or BUILDING, the corresponding bin is decremented by the necessary cost.

4.3.2 Resource Types

By default, JCiv MUST offer at least the following RESOURCES: wood, food, iron.

OPTIONAL RULE At the designers’ discretion, JCiv MAY also offer additional RESOURCE types. There MUST, however, be some mechanism to acquire and spend these novel RESOURCES. All such extensions MUST be documented in the player’s guide.

4.3.3 Extracting Resources

A player extracts RESOURCES from CELLS by assigning UNITS to that cell with the Produce COMMAND. Each UNIT can produce at most one FRU of the TERRAIN TYPE’s RESOURCE per production turn. Each TERRAIN TYPE has a *resource limit* that determines how many UNITS can be Produceing on that cell simultaneously. (A cell may *contain* an arbitrary number of UNITS simultaneously, but only a limited number of them can be Produceing.)

In addition, each TERRAIN TYPE has a *production time* that specifies how long (in GLOBAL CLOCK units) it takes a single UNIT to Produce a single FRU of that CELL’s RESOURCE. Thus, it is “easier” to produce food in the plains than it is to produce iron in the mountains.

4.3.4 Spending Resources

RESOURCES are consumed in two ways: actively (through the action of COMMANDs) and passively (to maintain UNITS and BUILDINGs). Both are tracked by scheduling an event in the GLOBAL CLOCK schedule. When the event is encountered and executed, the corresponding number and proper types of RESOURCES are deducted from the player’s BANK. For example, suppose a Consume(u_{97}) event is scheduled for GLOBAL TIME 972. When the GLOBAL CLOCK reaches 972, the Consume event is removed and one unit of food (the maintenance cost of a standard UNIT) is deducted from the BANK. Because Consume is a repeating event (so long as the UNIT is alive), a new Consume event is scheduled for

The cost for each COMMAND is listed in Tables 3 and 5. The maintenance cost for UNITS and BUILDINGs is listed in Tables 2 and 4, respectively.

When an event is encountered, if the player's BANK does not have enough of the appropriate type(s) of RESOURCE(s) to cover it, the event fails. The result of failure depends on whether the event was active or passive:

Active When a COMMAND fails, the result of the COMMAND is simply not executed. RESOURCES *are*, however, consumed, up to the BANK's ability. E.g., suppose BUILDING b_{11} was attempting to Spawn a Soldier (cost: 1 food and 1 iron). Suppose the BANK currently has 4 food and 0 iron. The Spawn fails and no Soldier is produced, but 1 food is still consumed.

Passive When a maintenance cost fails, the corresponding UNIT or BUILDING is destroyed (starvation). RESOURCES are still consumed up to the BANK's ability to pay, however. For example, suppose that the maintenance event for Building b_7 (a Fortress – maintenance cost: 1 food, 1 iron) is not met, though the BANK has 0 food and 3 iron. The Fortress is destroyed and the BANK loses 1 iron.

OPTIONAL RULE At the designers' discretion, the starving UNIT or BUILDING may defect to another player, rather than being destroyed. The exact conditions of defection are up to the designers, but **MUST** be documented in the player's guide.

4.3.5 Capturing Resources

When a BUILDING is successfully Invaded, the invading player may capture some RESOURCES from the invaded player. For each type of RESOURCE specified in the "Capture val" column of Table 4, the game engine generates a random integer between 0 and the specified number. The resulting number of FRUs of that resource are deducted from the invaded player's BANK and added to the invading player's BANK.

For example, suppose that Alice owns a Fortress, and Bob successfully Invades that Fortress.² Bob will receive between 0 and 3 food and between 0 and 3 iron. Suppose the game engine generates the values 0 and 2. Then Alice loses 2 FRUs of iron from her BANK, and Bob's BANK gains those 2 FRUs of iron.

4.4 Units and Commands

Conquest and growth are carried out by a player's UNITS. UNITS represent groups of people such as teams of serfs or squads of soldiers. The actual number of people comprising a UNIT is irrelevant because the UNIT is moved as an atomic group. The important factor is what UNITS do: they can occupy territory, extract resources, build buildings, attack other units, defend themselves, and attack or invade buildings.

Every UNIT is owned by a single player. An arbitrary number of UNITS belonging to a single PLAYER can occupy a single CELL at a time, but UNITS owned by two different PLAYERS cannot occupy the same CELL at the same time.

4.4.1 Unit Types and Properties

The JCiv implementation **MUST** support at least three types of UNITS: Serf, Soldier, and Scout. The units are differentiated by their properties. All UNITS have the following properties:

²The evil, aggressive, imperialist son of a marmot.

Unit Type	Mv	Vis	Att	Def	Build	Spawn Cost	Maint. Cost	Maint. Time
Serf	8	3	1	1	40	1 food	1 food	50
Soldier	5	3	5	3	70	1 food; 1 iron	1 food	35
Scout	3	8	2	2	80	2 food	1 food	40

Table 2: Unit types.

Move rate (Mv) This number is a *multiplier* for the TERRAIN TYPE movement cost value given in Table 1. The product of the UNIT’s base move and the TERRAIN TYPE’s movement cost gives the number of GLOBAL CLOCK ticks that it takes that UNIT to *enter* a CELL of that TERRAIN TYPE. E.g., it takes a Serf 16 (=8*2) clock ticks to enter a forest CELL, while it only takes a Scout 6 (=3*2) clock ticks to enter that same CELL.

Vision (Vis) This specifies how far that UNIT can see, in CELLS. The TERRAIN TYPEs and contents (BUILDINGs and UNITs) of all CELLS within the vision radius of that UNIT are displayed to that UNIT’s player. The vision property is a radius, measured in the Manhattan metric.

Attack (Att) This is the offensive value for this UNIT when executing an Attack COMMAND. This value is halved (rounded up) when the UNIT is being used to Invade a BUILDING. See Section 4.6 for details on this property.

Defense (Def) This is the base defensive value for this UNIT when it is being attacked. This value is doubled when the UNIT is executing a Defend COMMAND. See Section 4.6 for details on this property.

Build Rate (Build) The amount of time it takes a UNIT of this type to Build any sort of building.

Spawn Cost This is the RESOURCE cost to Spawn a UNIT of this type. This cost is paid when the Spawn command executes.

Maintenance Cost (Maint. Cost) The cost to maintain a UNIT of this type. This cost is paid every “Maintenance Time” ticks of the GLOBAL CLOCK.

Maintenance Time (Maint. Time) How often the Maintenance Cost for a UNIT of this type must be paid, in ticks of the GLOBAL CLOCK.

The values of the properties for different UNIT types are given in Table 2.

OPTIONAL RULE Additional UNIT properties MAY be implemented at the designers’ choice to support more and varied types of UNITs. While the base UNITs specified here MUST obey the property definitions given above, novel UNIT types MAY treat them differently. E.g., a SwampRat unit might pay a different movement cost for TERRAIN TYPE Swamp than for TERRAIN TYPE Mountain.

4.4.2 Unit Commands

All UNITS can execute the COMMANDs listed in Table 3. To execute a command, the player selects one or more UNITS and directs them to apply the command, optionally with a target CELL, UNIT, or BUILDING. The UI MUST provide a reasonably convenient way to do this selection and assignment. For example, the UI might allow the player to select a single CELL and then provide a checkbox-list of UNITS occupying that CELL. The player would then select a set of UNITS by picking from the checkboxes. The UI would offer a list of possible COMMAND (via buttons or a pick-list) and the player would pick a COMMAND. That COMMAND would apply to all selected UNITS. If the COMMAND requires a target, the UI would prompt the player to pick a target CELL, BUILDING, or UNIT. Finally, when the UI has all the information about the COMMAND, it would schedule execution of the COMMAND.

Command	Recur	Length	Target	Description
Produce	yes	Tab 1	occupied CELL	Produce one FRU of the RESOURCE for the occupied CELL's TERRAIN TYPE
Move	yes	Sec 4.4.1	adjacent CELL	UNIT moves into the adjacent CELL
Build	no	Tab 2	bldg. type	Construct a building of the specified type on the occupied CELL
Attack	yes	base move	UNIT or BUILDING	Attack and destroy a UNIT or BUILDING in an adjacent CELL
Defend	yes	instant	none	Stand and defend; take no other action and do not move.
Invade	yes	base move	BUILDING	Invade and occupy a BUILDING in an adjacent CELL without destroying it

Table 3: Commands available to UNITS.

The properties of a COMMAND are:

Recur Whether this COMMAND recurs automatically or not. If “yes”, then as soon as the UNIT completes this COMMAND, it automatically re-executes the command and schedules a new event for it. This process continues until the player assigns the UNIT a new COMMAND or the UNIT is destroyed. If “no”, then the UNIT becomes idle at the end of the COMMAND and takes no other action until the player explicitly directs it to.

Length How long it takes to execute this COMMAND, measured in ticks of the GLOBAL CLOCK.

Target What other thing this COMMAND affects, if any. Many times, the target will have to be explicitly specified by the player, but some times it can be inferred automatically. E.g., the Produce COMMAND can only apply to the CELL that the UNIT is occupying.

Description What the COMMAND does.

Note: The UI MUST NOT allow the player to select an illegal action. The following are illegal:

1. It is illegal for a player's UNITS to Attack or Invade his or her own UNITS or BUILDINGS.

Bldg. Type	Att.	Def.	Build Cost	M. Cost	M. Time	Capture Val
Town	1	1	1 food; 2 wood	1 food; 1 wood	80	5 food; 3 wood
Fortress	5	3	1 food; 1 wood; 1 iron	1 food; 2 iron	75	3 food; 3 iron

Table 4: Building types.

2. It is illegal for a UNIT to Attack or Invade a CELL not immediately adjacent to the UNIT. The UNIT MUST Move up to adjacent to the CELL in question and then invoke the Attack command.

OPTIONAL RULE The designers MAY extend the Attack and Invade commands to support “Move-then-attack” semantics.

4.4.3 Movement and Path Planning

Because it would be incredibly tedious to have to specify *every* CELL of movement for *every* UNIT, Jciv supports long-range movement planning. Specifically, when the player selects one or more UNITS, and gives them a Move command, the player may select *any cell* (including a currently invisible cell) as the target of that Move. The UNIT is responsible for translating that Move into a series of low-level, CELL-by-CELL Move commands.

The UNIT must, therefore, decide on a route to take between its current and target locations. The UNIT MUST take the *minimum cost* (smallest time) path between its current and target locations. To determine that path, the UNIT MUST use a shortest-path algorithm such as Dijkstra’s algorithm. A number of references to Dijkstra’s algorithm are available, including on the web and in Cormen, Leiserson, Rivest, and Stein’s text, *Introduction to Algorithms*. There will be a class lecture on shortest-path algorithms.

The UNIT MUST only plan its path with respect to the visibility information (Section 4.2.2) that is available at the time that the COMMAND is issued. The UNIT MUST NOT use the TERRAIN TYPES for any CELLS that are currently invisible to the player. It MAY treat invisible CELLS as having either minimum or maximum cost to traverse, at the designers’ option.

4.5 Buildings

Any CELL may contain 0 or 1 BUILDING. A BUILDING is owned by a player and assists that PLAYER by providing RESOURCE income and DEFENSE. In addition, new UNITS can only be Spawnd at a BUILDING.

4.5.1 Building Types and Properties

Table 4 gives the types of buildings that Jciv MUST support. All BUILDINGS have the following properties:

Attack (Att) The attack value of this BUILDING. All BUILDINGS can attack CELLS within 2 squares of them. See Section 4.6 for details on attacks by BUILDINGS.

Command	Recur	Length	Target	Description
Produce	yes	Tab 1	adjacent CELLS	Produce one FRU of the RESOURCE for the CELL occupied by the BUILDING and for every adjacent cell. Note: The Produce event should be scheduled separately for each affected cell.
Defend	yes	instant	none	Defend the BUILDING from attacks. In addition, while Defending, the BUILDING adds its defense value to the defense values of any UNITS in any adjacent CELLS.
Attack	yes	8	UNIT or BUILDING	Attack a UNIT or BUILDING within a radius of 2 CELLS of this BUILDING.
Spawn	no	25	UNIT type	Create a new UNIT. The new UNIT begins at the CELL occupied by the BUILDING.

Table 5: Commands available to BUILDINGS.

Defense (Def) The defense value of this BUILDING when attacked. This value is doubled when a BUILDING is assigned to Defend. See Section 4.6 for details.

Build Cost RESOURCE cost to create a BUILDING of this type, paid on completion.

Maintenance Cost (M. Cost) RESOURCE cost to maintain this BUILDING, paid every “Maintenance time” ticks of the GLOBAL CLOCK.

Maintenance Time (M. Time) How often the “Maintenance cost” of this BUILDING must be paid, in ticks of the GLOBAL CLOCK.

Capture Value (Capture Val) The potential RESOURCES that can be acquired by Invadeing this BUILDING. See Section 4.3.5 for details on capturing resources.

All BUILDINGS have a vision radius of 3 CELLS.

All BUILDINGS have 3 “damage points”. That is, a BUILDING must be successfully Attacked 3 times to be destroyed, or successfully Invaded 3 times to be occupied.

OPTIONAL RULE As with UNITS, additional BUILDING types with different or novel properties can be offered.

4.5.2 Building Commands

Just like UNITS, BUILDINGS can execute commands, as specified in Table 5.

The properties of BUILDING COMMANDS are the same as for UNIT COMMANDS (Section 4.4.2). Like UNIT COMMANDS, the UI MUST offer a way to select BUILDINGS, targets, and COMMANDS, and must prevent the player from executing illegal COMMANDS.

4.6 Combat

When one UNIT Attacks another, or a UNIT Attacks or Invades a BUILDING, or a BUILDING Attacks a UNIT or another BUILDING, a combat ensues. All combat has the same basic mechanic: both attacker and defender generate a random integer between 0 and 9 and add their attack/defense values, respectively. The higher total wins, with defender winning ties. The result depends on the type of combat:

4.6.1 UNIT-to-UNIT

If both UNITS are set to Attack, then the loser is destroyed. If one UNIT is set to Defend (or anything except Attack) then, if it wins, the other UNIT is unharmed. If it loses, it is still destroyed. If both UNITS are set to Defend (or anything except Attack), then no combat occurs.

4.6.2 UNIT-to-BUILDING

If the BUILDING is set to Attack and it wins, then the UNIT is destroyed. If the BUILDING is set to Defend (or anything except Attack) and it wins, then the UNIT is unharmed. If the UNIT is set to Attack, then the BUILDING is damaged by one damage point. When the BUILDING's damage points are exhausted, the BUILDING is destroyed. Note that a BUILDING can Attack UNITS within 2 CELLS of it, while a UNIT must be adjacent to Attack the BUILDING.

4.6.3 Invading

When a UNIT Invades a BUILDING, it still does damage points to the BUILDING, but at the end, the BUILDING is not destroyed. Instead, the BUILDING becomes the property of the Invading force. Any other UNITS in that BUILDING are destroyed, and the Invading UNIT moves into the BUILDING. The Invader captures RESOURCES associated with that BUILDING (Section 4.3.5), and the BUILDING's maintenance counter is reset.

4.6.4 BUILDING-to-BUILDING

The same Attack vs. Defend semantics hold for this case, and each BUILDING still has 3 damage points. A BUILDING cannot Invade another BUILDING, however.

OPTIONAL RULE The designers are free to implement additional combat mechanics and rules.

4.7 Victory Condition

The game ends when one of two conditions occur:

1. Only a single player remains. All of the UNITS and BUILDINGS of every other player have been destroyed or captured.
2. The fixed time-limit (t_{\max}) for the game expires.

s_X	Size of game grid in the X (left-to-right or east/west) direction.
s_Y	Size of game grid in the Y (top-to-bottom or north/south) direction.
t_s	Game speed (milliseconds). Wall-clock delay between increments of the global game clock.
t_{\max}	Game length (seconds). Wall-clock time determining the total length of game play.

Table 6: Summary of game parameters.

4.8 Game Parameters

Parameters that affect the game configuration and play are summarized in Table 6

5 Network J Civ

The J Civ game is to be played over the network with a single server and one or more clients. The server maintains the game engine, GLOBAL CLOCK, and game state. The clients maintain each player's UNITS, BUILDINGS, BANK, and information about the visible MAP. In addition, the client maintains the player's UI, allowing the player to see state of the game, manipulate UNITS, issue COMMANDS, etc.

The server MUST ensure proper synchronization between clients and ensure that all events are atomic, no clients corrupt any other's data, etc. The server MUST NOT allow the game state to become corrupted, regardless of the number of clients or the load.

OPTIONAL RULE The designers may, at their option, offer a single-player mode in which the computer plays one or more opponents against the single player.

6 Requirements

6.1 User Interface

The J Civ program MUST include at least one client program that provides a visualization of the game state at all time points. The program team MAY also provide additional support programs, at their discretion. For example, a board editor or a debugging interface that displays the complete game state at any time would be useful tools.

The J Civ client program MUST provide at least the following user interface elements. It MAY also provide additional UI functionality. The specific ways in which the UI is implemented is up to the design group, but part of the evaluation of the program will be on the quality and intuitiveness of the user interface.

1. Ability to load/store a MAP configuration.
2. Ability to specify the name of a player.

3. Ability to choose a server address and port number.
4. Ability to single-step the game state or run in free-running mode. (This ability may be provided through an interface to the server or through special commands issued by privileged clients.)
5. Visualization of the visible part of the MAP (Section 4.2.2), including the TERRAIN TYPEs of visible CELLS and UNIT(s)/BUILDING(s) in those CELLS.
6. Ability to find the name of the player who owns a given UNIT or BUILDING (e.g., by clicking on that CELL or hovering over it.)
7. Ability to scroll to all parts of the MAP, even if parts of it are currently invisible.
8. Ability to select any/all UNITS in a single CELL and apply a single COMMAND to all selected UNITS.
9. Ability to select a BUILDING and apply a COMMAND to that BUILDING.
10. Current contents of the player's BANK (number of each type of RESOURCE in the BANK).
11. Number of each type of RESOURCE that will be required by a selected COMMAND.
12. A warning to the player when the BANK may not have enough resources to complete a requested COMMAND. (This should not prevent the COMMAND from being issued, but should allow the player to change her/his mind.)
13. Notification to the player whenever a UNIT or BUILDING is under Attack, Invade, or is destroyed or captured.
14. Notification to the player whenever a COMMAND fails because of insufficient resources.
15. Notification to all players whenever another player enters or leaves the game (including when a player is defeated).

The program MAY support a configuration file format to load some of the game configuration but it MUST also support some way to enter that data through the user interface.

The program(s) MAY provide command line switches to modify its behavior as well. All such switches MUST be documented in the user documentation.

6.2 Multithreading

The Jciv program MUST provide a multithreaded server for the networked game. That is, the network server MUST dedicate (at least) one thread to each player client program. The threads must cooperate to update the centralized game state maintained on the server. The threads MUST be correctly synchronized so that no game state is corrupted. The design of the multithreaded component MUST be described in design and final performance documentation. The server MUST NOT spawn more threads than are necessary, it MUST clean up threads that have terminated (e.g., because a client has left the game), etc. In particular, there must never be more than $O(n)$ threads running for a game with n clients (including both clients and server).

7 Deliverables

This section describes the content to be delivered during the project. Unlike previous projects, the actual milestones in this project are to be specified by the design group. The first deliverable is the development schedule along with milestones. *The team will be held to the schedule they provide in Milestone 1.*

7.1 Milestone 1: Development Plan

The first part of this project is a planning phase. The development group **MUST** meet among themselves soon after the project is released and work out a full development plan including modular decomposition, testing plan, group member responsibilities, timeline (including, but not limited to, intermediate milestones), and intermediate deliverables. The design group is to schedule time with the **CLIENTS on or before 3:00 PM, Apr 11** to deliver and discuss the development plan and schedule. The initial design documents are to be provided hardcopy, in duplicate, and signed off by the design group and **CLIENTS** during the meeting.

The timeline **MUST** include at least 2 intermediate milestones. Each milestone **MUST** specify a delivery date and time as well as the content due. Specific content for each milestone is up to the design group, but **MAY** include items such as design documentation, class hierarchies, interfaces, code, test suites (unit, regression, and/or integration), working prototypes, API documentation, user documentation, demonstrations, etc.

The design group will be held to and graded on their adherence to their development schedule. Modifications to the development schedule are permitted, *so long as the modification is requested and approved at least three days in advance of the milestone delivery date, a new date is proposed and approved, and a new set of deliverables are proposed for the revised milestone.*

In addition to intermediate milestones, each group is to schedule a weekly, half-hour progress meeting with the **CLIENTS**. (This meeting may be included with the milestone meeting during the weeks of milestone deliveries.) Other than milestones, there need be no specific deliverables for such progress meetings, but the group members **MUST** be able to demonstrate progress during the week and account for time spent.

Dates and times of meetings and intermediate milestone delivery **MUST** be coordinated with the **CLIENTS** (specifically, with Prof. Lane). Feel free to see Prof. Lane during office hours before April 11 to work out acceptable meeting times. Meeting times may have to be rescheduled dynamically as Prof. Lane's other commitments impinge, but he will make a strong effort to stick to the original schedule.

7.1.1 Milestone Blackout Dates

Prof. Lane will be unavailable Apr 21–23. No milestone deliveries may be scheduled then.

7.2 Rollout

The final delivery of the full project is to be scheduled in the Development Plan, but delivery **MUST** be between 9:00 AM on Mon May 9, and noon on Fri, May 13. **NOTE:** The drop-dead date for this project is noon, Fri, May 13. *NO Project deliveries will be accepted after this time.*

In addition to deliverable content (listed below), rollout will include a final delivery meeting with the CLIENTS. At this time, the group must demonstrate the full functionality of its project, provide a walk-through of usage, demonstrate example test code, etc.

Deliverable content for this project includes:

JCiv.jar A jar archive file containing all class files necessary to run the JCiv program(s) and provide all specified functionality. Optionally, the designers may provide a separate jar archive for the client and for the server.

Java source files A subdirectory named `src/` MUST be provided that includes the entire Java source code tree for all `.class` files in the `JCiv.jar` file, as well as any other support code (e.g., programs used in testing the code that are not part of the JCiv program itself). *Note:* if these programs depend on external library code other than the Java JDK or the `gnu.getopt` suite, the submission tarball MUST either include the library whole or provide easy and explicit instructions on how and where to access such libraries. This documentation MUST be provided in the `README.TXT` file. The designer is responsible for ensuring that all copyright and distribution conditions are adhered to.

README.TXT This file MUST describe how to compile, configure, and install the JCiv program suite, including how to construct the `.jar` file from the raw source code. It MUST also list any dependencies on additional software support libraries. Furthermore, if the software suite provides more than one executable (e.g., a MAP editor or graphical debugger is provided), the `README.TXT` file should indicate what programs are included in the suite and how to run them. Complete documentation for such auxiliary programs MUST be included in the user documentation.

Internal documentation The handin MUST also include the full, compiled JavaDoc documentation for all Java source files in the submission tarball. This documentation MUST include full descriptions of every public or protected method, field, sub-class, enclosed class, or constructor employed by the code. This documentation hierarchy MUST be included in a sub-directory named `documentation/` within the submission tarball package.

Player/User documentation The handin submission MUST include complete player (user-level) documentation for the JCiv program suite. This documentation MUST include descriptions of how to use the main game client program as well as any auxiliary programs (e.g., a separate MAP editor). It must describe the User Interface of the program, how to interpret output/buttons/dialogs/screens/etc., and at least one step-by-step example of running the program, playing a game, interpreting the results, etc. In particular, this document MUST explicitly describe *both* behavior required by this specification *and* any extensions or modifications provided by designers (e.g., **OPTIONAL RULES**).

This document SHOULD describe both client operation and behavior and server operation, configuration, and behavior. The designers MAY, however, choose to offer a separate administrator's guide that describes how to run and configure the server. In either case, there MUST be clear and distinct directions on how to install, run, and configure the server.

This document **MUST** be named `PLAYERGUIDE.extension`, but it **MAY** be a plain text, HTML, PDF, or PostScript document (with the appropriate `extension`). It **MUST NOT** be a Microsoft Word or other nonportable format document.

Design documentation The final submission must also include one or more documents that describe both the initial design of the system (as proposed in Milestone 1), the final design and features of the system (as delivered in the Rollout), and a rationale justifying the modifications (if any). **Note:** Modifications to the original design are not bad. In fact, they are expected and encouraged. But the design documentation **MUST** describe *why* any such changes are made. E.g., refined understanding of the specification, design discussions with the **CLIENTS**, improved efficiency or modularity, improved use of Java language features, scaling back design to meet deadlines, untenable original design, improved support for requirements, etc. are typical reasons for modifying the design.

Test cases and documentation The submission tarball **MUST** include a subdirectory named `tests/` that includes all of the data or test cases used in unit, regression, or integration testing of the `JCiv` system. In addition, the submission **MUST** include a document named `TESTING.extension` that describes the testing methodology, result of test cases, etc. This document is responsible for convincing the **CLIENTS** that the `JCiv` suite performs correctly and meets this specification document. This document **MAY** be a plain text, HTML, PDF, or PostScript document (with the appropriate `extension`). It **MUST NOT** be a Microsoft Word or other nonportable format document.

CVS log file(s) For each Java source file, the submission tarball **MUST** include a corresponding `.log` file including the CVS log for that sourcecode.

Team evaluations Each team member must independently provide a written assessment of their own performance within the design team as well as the performance of the other team members. This is to be delivered by email directly to the **CLIENTS** and is not to be included in the group submission package. This information will be kept strictly confidential by the **CLIENTS**. See Section 2.1 for details on this document.

8 Grading

The grade for this project is a group grade — each individual will receive credit for the performance of their team’s entire project. It behooves the team members to work hard and support each other to provide the best possible project.

The following is a *rough* guide to the credit associated with various levels of implementation. These guidelines are *not* absolute and do not guarantee any specific grade. Superficial achievement of the requirements for one level may still yield a lower grade if, e.g., code is poorly designed, coded, or tested or if documentation is incomplete, poorly written, or incoherent. “C” level is considered to be the minimal “functional” level of implementation.

C level Client/server functionality with correct event scheduling and synchronization. Client user interface including the ability to see different terrain types and ability to see and manipulate

individual UNITS and BUILDINGS. UNITS do path planning with shortest Manhattan distance (i.e., smallest number of CELLS, rather than smallest time.) No handling of visibility. Full player (user) documentation, API documentation, CVS logs, and test/validation results.

B level All of C level, plus: ability to manipulate *groups* of UNITS. Units do path planning using correct, shortest-path distance (via, e.g., Dijkstra's algorithm).

A level All required (MUST) functionality, including handling visibility, dynamic loading and saving of MAPs, and all other specified functionality.

A+ level All required (MUST) functionality, plus at least one *significant* extension to the specified functionality. (Adding a single new TERRAIN TYPE does not constitute significant functionality. To attain A+ level credit, the group will have to demonstrate one or more features that require significant design effort and creativity above and beyond that required by this document.)

9 Timeline

Apr 4, 3:00 PM Project description handed out

On or before Apr 11, 3:00PM Milestone 1 due

May 9-13 Rollout and delivery meeting (to be scheduled by group)

TBA Final exam

May 13, noon Drop dead date for P3 rollout

May 13, noon-2:00 Class demos³

All times are given in the Mountain time zone.

³If both groups complete their projects satisfactorily, we will have a class play-off and Prof. Lane will spring for lunch during the demos.