**COMP 361 Projet**

**Medieval Warfare**

**Requirements Models**

Team HHDP

Parker Dinh-Williams 260 479 894

Justin Domingue 260 588 454

Harry Simmods 260 502 863

Daniel Pham 260 526 252

**1. Uses Cases**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Use Case:** Play an entire game

**Level:** Summary

**Intention in Context:** The intention of the Users is to play an entire game by first authenticating to the server and then playing turn by turn until the game is over.

**Multiplicity:** Three users play together

**Primary Actor:** User (becomes Player, once authenticated through the System)

**Main Success Scenario:**

1. AuthenticatePlayer
2. StartMultiplayerGame

While the game is not over,

1. Each *Player* Play[s]Turn

Then,

1. System End[s]Game

**Extensions:**

3a. *Player* forfeit to the *System* and quits the game. If there are still at least two Players in the system, *System* removes *Player* from Game and goes to 2. Otherwise, go to 3.

3b. *Players* leaves the game while playing his turn (without forfeiting); All the actions he performed since the start of his turn are discarded. *Player* will have to load the game.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**User Case**: AuthenticatePlayer

**Primary Actor**: User

**Secondary Actor**: GameCenter

**Intention**: The intention of the User is to authenticate to the System.

**Level**: User Goal

**Multiplicity**: One *User* will authenticate to the GameCenter

**Main Success Scenario**:

1. *System* asks the *GameCenter* to start the authentication process
2. *Player* provides *GameCenter* with required login information
3. *GameCenter* validates the input and communicate the authentication success or failure with the *System*
4. *GameCenter* cannot validate the *Player* input.

*If the authentication failed, Go back to 2 (the use case cannot continue until the user has successfully authenticate).*

 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Use Case:** StartMultiplayerGame

**Primary Actor:** User

**Secondary Actor:** GameCenter

**Intention:** The intention of the User is to start a multiplayer game with two other Users.

**Level:** User Goal

**Multiplicity:** Three *Users* will connect to start a game.

**Main Success Scenario:**

1. *Player* starts a new game through the *System*
2. *System* requests opponents from *GameCenter*
3. *System* assigns the opponents to the *Player*s
4. Each *Player* Select[s]Map
5. *System* acknowledges *Players*' map selections and arbitrarily chooses one
6. *System* starts the game

**Extensions:**

2a. *Player* loads a game in progress - go to step 7

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Use Case:** SelectMap

**Primary Actor:** Player

**Intention:** Player wants to select a map.

**Level:** Subfunction

**Main Success Scenario:**

1. *System* displays a list of maps to the *Player*
2. *Player* informs the selection to the System

**Extensions:**

2a. *Player* doesn't perform his selection before the timer runs out - the *System* chooses a random map.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Use Case:** PlayTurn

**Primary Actor:** Player

**Intention:** The intention of the Player is to play a turn according to the rules.

**Level:** User Goal

**Multiplicity:** One player will play a turn at a time. The other players cannot perform any state-changing actions.

**Main Success Scenario:**

1. *Player* Go[es]ThroughPassivePhases
2. *Player* cycles through his enabled units; for each unit *Player* Perform[s]ActionOnUnit
3. *Player* notifies the System he is done with his turn

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Use Case**: GoThroughPassivePhases

**Primary actor:** Player

**Intention:** Player wants to go through the passive phases (tree growth, tombstone, build, income, payment and death phase) collecting knowledge on the current game state.

**Level:** Subfunction

**Main Success Scenario:**

1. *System* presents the phases to the *Player*
2. *Player* navigates through them

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Use Case**: PerformActionOnUnit

**Primary Actor**: Player

**Intention:** The intention of the Player is to perform actions on one of his/her unit according to the rules of the game.

**Level:** Subfunction

**Main Success Scenario:** (consider a "move" action)

1. *Player* selects one of his units
2. *System* visually highlights the unit to indicate it is selected
3. *System* tells him/her possible (allowed) move locations
4. *Player* asks the *System* to move his/her unit to one of those locations

(Use Case is similar for the other actions like *upgrade*, *combine*, *build*, *recruit*, *build*, etc.)

1. *System* disables unit

**Extensions**

4a. *Player* changes his mind and wants to cancel the selection on the current unit by selecting another tile. Go to 2.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Use Case**: EndGame

**Primary Actor**: System

**Intention:** The intention of the System is to end the current game and display the game information.

**Level:** User goal

**Main success Scenario:**

1. *System* notifies users that the game is over
2. (*System* updates player rankings)
3. *(System* displays player rankings)
4. *System* closes, deletes game state and sends players back main menu

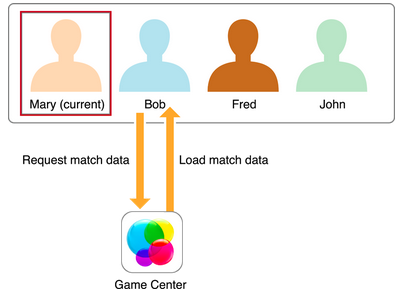
**2. Architectural Decision**

Client-Server model through Apple’s Game Center.

The server and its API are provided through GameKit so we only need to develop a client.

How Game Center servers work for turn based games:

* Upon initialization, the server holds a centralized master copy of the game state.
* Only one player is allowed to update that master state
* As the player takes actions, the game updates the match data and transmits it back to Game Center.
* Inactive players can query the server to obtain the current game state.
* All of the game functionality will reside on the client side as the server can pass data but cannot act on it. (Trust that the user only performs legal action on the most up to date state of the game)
* From GameKit’s documentation:



Matchmaking:

* The game presents the standard Game Center matchmaking user interface where they must login. This interface allows the users to create a room and fills the remaining slots with other players searching for a match.
* The client provides a way for the user to search for a match wishing to fill their empty slots.

Leaderboards:

* When a match ends, the client reports the outcome of the game to Game Center so that it can be stored.

**3. Requirements Models**

**3.1 Structural Requirements**

**3.1.1 Environment Model**

**Input Messages (:ClientGUI -> :GameEngine)**

**newGame(player: Player)** - sent by ClientGUI to initialize new game

**selectMap(name: String)** - sent by ClientGUI to choose map selection within GameEngine

**loadGame(game: Game)** - sent by ClientGUI to load previous game

**move(from: Tile , to: Tile)** - sent by ClientGUI to move selected unit from one Tile to another

**upgradeUnit(selected: Tile , toUnitType: Unit)** - sent by ClientGUI to upgrade selected tile unit type

**upgradeVillage(selected: Tile, village:VillageType)** - sent by ClientGUI to upgrade selected tile village type

**build(builder: Tile, on:Tile,  type: LandType)** - sent by ClientGUI to build selected tile to land type

**combine(source: Tile, with: Tile)** - sent by ClientGUI to combine source tile unit with another tile unit

**recruit(village: Tile, unit: UnitType)** - sent by ClientGUI to recruit selected tile w/ village to Unit type

**forfeit(Player : player)** - sent by ClientGUI for player to forfeit and lose the game

**endTurn()** - sent by ClientGUi once player is finished with turn

**Output Messages (:GameEngine -> :GameCentre)**

**authenticate(player: Player)** - sent to GameCentre to authenticate player sent once time triggered event goes off

**matchPlayers(game: Game)** - sent to GameCentre to match / find players for the selected game

**fetchGame(game: Game)** - sent to GameCentre to load / fetch the current game selected - will ask for updated game state

**pushState(game: Game)** - sent to GameCentre to upload the current game state once player has finished turn / selected map

**endGame(game: Game)** - sent to GameCentre to end the game due to a winner

**Input Messages (:GameCentre -> : GameEngine)**

**matchedPlayers(players: List<Player>, game: Game)** - sent by GameCentre to return players included in current game

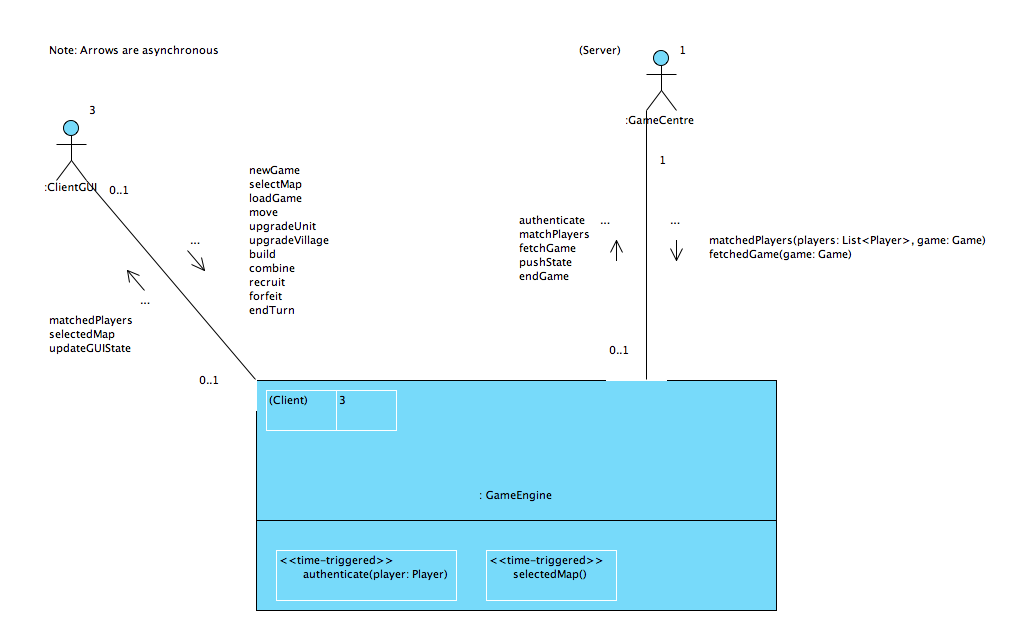
**fetchedGame(game: Game)** - sent by GameCentre to update the game state, or a fetched / loaded game

**Output Messages(:GameEngine -> :ClientGUI)**

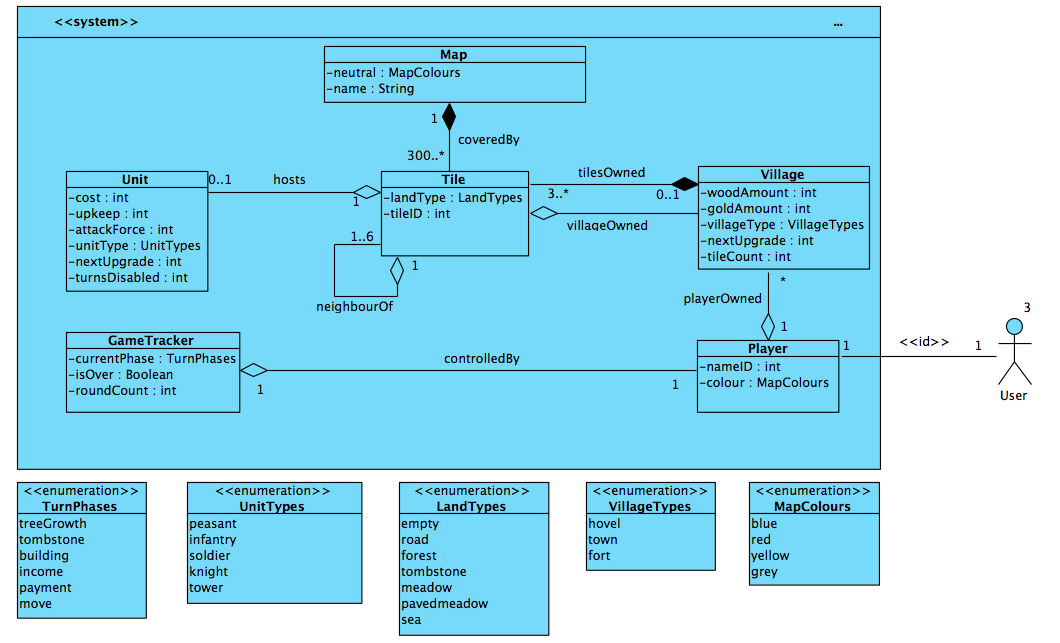
**matchedPlayers(players: List<Player>)** - sent to ClientGUI

**selectedMap(map: Map)** - sent to ClientGUI to show the map chosen for the game

**updateGUIState(game: Game)** - sent to ClientGUI to update the GUI state and update menus

****

**3.1.2 Concept Model**

****

**3.2 Behavioural Requirements**

**3.2.1 Operation Model**

**From GUIClient**

---------------------------------

**Operation**: System::newGame()

**Description**: Creates a new game.

**Post:** Inform GameCenter that the “host” want to create a new room. GameCenter then assigns 2 random players to the GameEngine.

**Scope**: GameEngine, Player.

**Messages**: GUI::{update}, GameCenter::{requestPlayers}

**Operation**: System::selectMap(name: String)

**Description:** Stores the player’s preferred map choice.

**Post**: Add the player’s prefered map name to the pool for selection and update the new state in GameCenter.

**Scope**: GameEngine

**Messages**: GUI::{update}, GameCenter::{pushState}

**Operation**: System::getGame(game: Game)

**Description**: Sends a request to Game Center to retrieve the matchData.

**Scope**: GameCenter

**Messages**: GameCenter::{getGame}

**Operation**: System::move(from: Tile, to: Tile)

**Description:** Moves a unit between 2 tiles.

**Pre:** Requires this to be a valid move (follow the game’s invading rules and nothing is blocking the passage).

**Post**: Removes the unit of the “from” tile and recreates him in the “to” tile. Find the set of tiles in the shortest path and update the affected tiles accordingly (trampling of meadows etc.).

Updates the “to” tile according to it’s content (removes trees and graves,).

If ‘to’ tile contains a village, take control of and don’t move the unit.

Disables the unit only upon invasion.

**Scope**: Map, Tile, Unit.

**Messages**: GUI::{update}

**New:** unit: Unit;

**Operation**: System::upgradeUnit(selected: Tile, to: UnitType)

D**escription:** Upgrades a unit’s UnitType.

**Pre:** Can only be performed if sufficient funds allow for it, if the village can manage with the increased upkeep and ‘to’ UnitType is greater than that of the current ‘selected’ Tile.

**Post**: Changes the UnitType of the unit on the ‘selected’ tile to that of ‘to’ and removes the cost from the associated village’s treasury. Set the unit to the disabled state.

**Scope**: Tile, Village, Unit, UnitType.

**Messages**: GUI::{update}

**Operation:** System::upgradeVillage(selected: Tile, village: VillageType)

**Description:** Upgrades a village’s VillageType.

**Pre**: Can only be performed if sufficient funds allow for it and the ‘village’ VillageType is greater than that of the current ‘selected’ Tile.

**Post:**Upgrades the village on the currently selected tile to the type of ‘village’ and removes the cost from its treasury. Set the village in the disabled state.

**Scope:** Tile, Village, VillageType.

**Messages:**GUI::{update}

**Operation**: System::build(builder: Tile, on: Tile, land: LandType)

**Description:** Creates a “feature” on a Tile (meadows, roads).

**Pre:** Can only be performed if the village’s funds allow for it, the unit on the ‘builder’ tile is not disabled and respects the game’s rule (meadow can only upgrade an empty tile etc.).

**Post**: Changes the ‘on’ land type to that of ‘land’. Removes the cost from the associated village’s treasury and disables the unit on the ‘builder’ tile.

**Scope**: Tile, LandType, Village, Unit.

**Messages**: GUI::{update}

**Operation**: System::combine(source: Tile, with: Tile, type: UnitType)

**Description:** Merge 2 units to create a stronger one.

**Pre:** Can only be performed if sufficient funds allow for it, if the village can manage with the increased upkeep, ‘source’ is not disabled

**Post**: Removes the units from the ‘with’ tile. Change the UnitType of ‘source’ to the desired upgraded UnitType and set its state to disabled.

**Scope**: Tile, Unit, UnitType, Village.

**Messages**: GUI::{update}

**Operation**: System::recruit(recruiter: Tile, to: Tile, unit: UnitType)

**Description:** Creates a new unit.

**Pre**: Can only be performed if sufficient funds allow for it, if the village can manage with the increased upkeep and if the ‘to’ Tile is both empty and owned by the village ‘recruiter’.

**Post**: Adds the new unit on the ‘to’ tile with its state set to disabled. Remove cost from the village’s treasury.

**Scope**: Map, Tile, Unit, UnitType, Village.

**Messages**: GUI::{update}

**New:** unit: Unit

**Operation:** System::endTurn()

**Post:** System ends the current player’s ability to update the state. Passes this power to the the next player in the rotation and updates GameCenter’s master state.

**Scope:** GameCenter, GameEngine.

**Messages:** GUI::{update}, GameCenter::{pushState}

**Operation**: System::forfeit(player: Player)

**Description**: Removes the player from the match.

**Pre:** A player can only call this with himself as a parameter.

**Post:** The ‘player’ is removed from the rotation and GameEngine can no longer assign control to him. The ‘player’ villages and units are removed. His controlled tiles are set to neutral. The control of the game then passes to the next player in the rotation.

**Scope**: GameCenter, GameEngine, Village, Unit, Tile.

**Messages**: GUI::{update}, GameCenter{pushState} (will also call GameCenter{endGame} if only one player remains)

**From Game Center**

---------------------------------

**Operations:** System::matchedPlayer(players: List<Player>, game: Game)

**Descriptions:** Sets the player fields for the specified ‘game’.

**Scope:** GameEngine

**Messages:** GUI::{MatchedPlayer}

**New:** player: Player

**Operations:** System::fetchedGame(game: Game)

**Description:** Recreates a game from the game data.

**Scope:** GameEngine, Maps, Players, Villages, VillageType, Units, UnitTypes, Tiles, TileTypes.

**Messages:** GUI::{update}

**3.2.2 Protocol Model**