Santorini

Documentazione

Andrei Petrisor (1085993), Andrea Rusconi (1086646), Uriel Fumagalli (1085169), Antonio Radu (1085992)

Ingegneria del Software A.A. 2024/2025

Indice

Software Engineering Management

Project Plan - Santorini

1. Introduzione

L'obiettivo è quello di creare una versione digitale con GUI del gioco da tavolo Santorini, utilizzando i metodi di sviluppo e le regole dettate dall'ingegneria del software in maniera professionale e seguendo ciò che si è studiato nel corso.
Le persone responsabili del progetto sono:

- Andrei Petrisor (1085993)
- Andrea Rusconi (1086646)
- Uriel Fumagalli (1085169)
- Antonio Radu (1085992)

2. Modello di processo

Utilizzeremo il framework di sviluppo AGILE SCRUM. Organizzeremo lo SCRUM seguendo degli sprint di durata circa una settimana. Ogni giorno avremo un “daily scrum” utilizzando WhatsApp o in chiamata su Discord oppure avendo uno “stand up meeting” di persona quando possibile.

Pietre miliari:

a. Creazione vari UML
b. Implementazione classi e metodi con papyrus (Versione **alpha**)
c. Creazione parte server con GUI (Swing)
    i. Gestione connessioni client/server
d. Implementazione regole gioco base (da papyrus) sul server (Versione **beta**)
e. Aggiunta “god powers” semplici (Versione **1.0**)
f. Implementazione leaderboard
g. (Opzionale) Aggiunta “god powers” avanzati

3. Organizzazione del progetto

Il cliente domanda la versione digitale del gioco in scatola “Santorini”. Richiede che il programma possieda le regole complete del gioco in scatola e le prime due di queste difficoltà:

1. Gioco senza “god powers” (Difficoltà “Easy”)

2. Gioco con 10 “god powers” semplici (Difficoltà “Normal”)
3. Gioco con 20 “god powers” avanzati (Difficoltà “Advanced”). - opzionale

Inoltre il cliente vuole poter vedere una classifica nella pagina principale (Home) del gioco. - opzionale

Product owner: Uriel Fumagalli
SCRUM master: Andrei Petrisor
Developer: Uriel Fumagalli, Andrei Petrisor, Andrea Rusconi, Antonio Radu

4. Standard, linee guida, procedure

Data rilascio gioco: …
Linguaggio di programmazione: Java
Creazione GUI: Swing
Standard di programmazione: Oracle
Software creazione classi UML: Papyrus
IDE: Eclipse

5. Attività di gestione

Alla fine di ogni sprint avremo una riunione dove aggiorneremo lo stato dello sviluppo e analizzeremo il lavoro compiuto nello sprint (Sprint Review), insieme a una discussione sui possibili miglioramenti applicabili allo sviluppo del progetto (Sprint Retrospective).

Requisiti:

- Obbligatori
  - Gioco funzionante
  - Regole complete difficoltà “Normal”
  - 10 god powers semplici
- Opzionali
  - Aggiunta god powers avanzati
  - Aggiunta leaderboard

Tempi: è richiesta la consegna del programma funzionante entro 5 giorni dalla prova d’esame
Costi: I nostri costi non saranno monetari ma misurati in ore di lavoro per sviluppatore.

6. Rischi

I potenziali rischi potranno essere la mancanza di esperienza nello sviluppo e organizzazione del software, inoltre un potenziale rischio potrebbe essere la mancanza di ore di lavoro che ogni membro avrà da dedicare al progetto. L'hardware non sarà un potenziale rischio di questo progetto grazie alla leggerezza del software.

7. Personale

Competenze del team:
Andrei Petrisor, Andrea Rusconi, Uriel Fumagalli, Antonio Radu:
Undergraduate Junior Developer

8. Metodi e tecniche

Uso di MVC e Singleton pattern.
Testing: Il testing del codice è stato eseguito su tutti i metodi delle classi nel package "gameComponent" poco dopo la creazione degli stessi. Il testing della GUI è stato eseguito in concomitanza con la sua creazione.

Versione Alpha (gioco con solo difficoltà "Easy") - v0.1
Versione Beta (gioco con difficoltà "Easy" e "Normal") - v1.0

9. Garanzia di qualità

Riguardo ai fattori di qualità facciamo riferimento al modello di qualità esterno ed interno della standard iso 9126. Per quanto riguarda le organizzazioni e procedure per garantire i fattori di qualità si intende seguire il modus operandis delle organizzazioni SQA: monitorare adeguatamente il software durante il suo processo di sviluppo, assicurare il pieno rispetto degli standard e garantire che le inadeguatezze siano evidenziate affinché vengano sistemate dai developer.

10. Pacchetti di lavoro (workpackages)

Radu Antonio: prevalentemente documentazione, alcuni diagrammi (vedi github), aiuto interfaccia grafica.

Petrisor Andrei: prevalentemente interfaccia grafica, alcuni diagrammi (vedi github).
Fumagalli Uriel: prevalentemente backend, alcuni diagrammi (vedi github), testing.
Rusconi Andrea: aiuto GUI, alcuni diagrammi (vedi github), codice.

Nella fase iniziale, di progettazione, abbiamo lavorato a coppie per massimizzare le tempistiche, diminuire gli errori per rendere il tutto più efficiente.

11. Risorse

Qualsiasi dispositivo con accesso ad internet può giocare.
Fortemente consigliato l'utilizzo da Computer/Laptop.

12. Budget e programma

Il budget per questo progetto non sarà monetario ma definito da un quantitativo minimo di tempo utilizzato per lo sviluppo. Questo budget è identificato in 40 ore di lavoro minime per sviluppatore.

13. Cambiamenti

Utilizzando uno sviluppo AGILE del software la necessità di modifiche al progetto verrà gestita solo attraverso una riunione da parte di tutti i membri del team.

14. Consegna

La procedura di consegna per questo progetto é stata definita con una data specifica (5 giorni prima della consegna della presentazione) e tramite il caricamento su un repository github di una versione eseguibile del progetto, insieme a tutta la documentazione e tutti i diagrammi utilizzati per lo sviluppo del progetto.

Software Life Cycle

tipo di processo seguito

1) Product Backlog
2) Sprint Planning
3) Sprint
4) Daily Scrum: whatsapp e discord
5) Sprint Review
6) Sprint Retrospective

Differenze rispetto a quanto previsto nel project plan

Per mancanza di tempo si è deciso che la versione iniziale non sarà gestita con un client/server ma sarà semplicemente un'applicazione in cui i giocatori potranno giocare solo dallo stesso dispositivo. Possiamo considerarlo come un prototipo.

Sprint:
Sprint 1) dal 9 Nov 2024 al 18 Nov 2024 → Project Plan
Sprint 2) dal 18 Nov 2024 al 29 Nov 2024 → Diagrammi
Sprint 3) da 29 Nov 2024 al 13 Dic 2024 → Sistemare project plan e diagrammi
Pause festive e inizio sessione esami
Sprint 4) dal 23 Gen 2025 al 30 Gen 2024 → Prima versione del gioco, rifare diagrammi, finire documentazione, preprare presentazione.

Configuration Management

Strumenti utilizzati:

- Eclipse
- Github
- Kanban Board (su github)
- Swing
- Papyrus
- Draw.io
- Sonarqube
- Maven

People Management and Team Organization

Organizzazione del lavoro: Team SCRUM

Product owner: Uriel Fumagalli
SCRUM master: Andrei Petrisor
Developer: Uriel Fumagalli, Andrei Petrisor, Andrea Rusconi, Antonio Radu

Software Quality

Correttezza	L’utente richiede una copia digitale di un gioco in scatola
Affidabilità	Precisione del SW è data dalla modalità di gioco(Easy,Normal,Advance): più la difficoltà è elevata, più il SW è affidabile.
Efficienza	Le funzioni non sono troppo complicate, se non alcune regole o “god powers” particolari.
Usabilità	La praticità di gioco sta tutta nella GUI, la quale rende il SW molto più usabile, indicando in modo chiaro all’utente le varie possibilità/scelte di gioco.
Manutenibilità	La correzione di un errore non comporta un'enorme spesa di tempo in quanto il SW non è troppo grande, ed è abbastanza organizzato da poter trovare quasi immediatamente qualsiasi errore.
Testabilità	Testabilità poco complessa in quanto le possibilità dei giocatori sono limitate, unidirezionali e cicliche.
Flessibilità	Le fasi di gioco sono molto distinte il che rende il programma facilmente modificabile, permettendo future aggiunte senza troppi problemi.
Portabilità	Per poter giocare basta avere una copia del gioco, in quanto le partite e tutte le regole sono gestite dall’applicazione.
Riutilizzabilità	Il programma è abbastanza specifico per questa tipologia di SW, ovvero giochi, il che lo rende poco riutilizzabile.

Requirement Engineering

Introduzione

1.1 Obiettivo

L'obiettivo del progetto è quello di riproporre una copia, il più fedele possibile, del gioco in scatola "Santorini".

1.2 Scopo

Lo scopo del progetto è di poter giocare al gioco "Santorini" da un qualsiasi dispositivo (Windows), con i propri amici, o chiunque abbia una copia del gioco.

1.3 Definizioni, acronimi e abbreviazioni

...

1.4 Riferimenti

Nessun riferimento (è tutto presente in questo file).

1.5 Panoramica

La sezione 2.0 spiegherà nei suoi sottopunti la descrizione generale del progetto mentre la sezione 3.0 si occuperà dei requisiti specifici.

Descrizione generale

2.1 Prospettiva del prodotto

Il programma gestisce sia le singole partite(le regole, la logica di gioco, i tempi di attesa, i turni, ecc…) sia la parte grafica (GUI), la quale verrà meglio definita nella sezione 3.0.

2.2 Funzioni del prodotto

Il Programma presenta un'unica funzione: giocare a “Santorini”.

2.3 Caratteristiche dell’utente

In questa versione del programma, in modo da avere un’esperienza più lineare, si richiede che gli utenti conoscano le regole del gioco, le quali sono caricate nella cartella “rules” su github. Altrimenti l’esperienza potrebbe non risultare piacevole, soprattutto nelle difficoltà più alte.

2.4 Vincoli

L’utente è limitato al gioco e non può fare altro.

2.5 Presupposti e dipendenze

…

Requisiti specifici

3.1 Requisiti dell’interfaccia esterna

- ### 3.1.1 Interfaccia utente

Una prima pagina “menu” con due pulsanti, il pulsante “gioca” e il pulsante “esci”. Una seconda pagina "scelta nomi giocatori” con: un campo di compilazione, un bottone per confermare il nome, un bottone per avviare la partita ed un bottone per tornare alla pagina “menu”. Una terza pagina “partita” con il campo di gioco, che consiste in un quadrato 5x5, nella quale è possibile: muovere i propri worker, costruire piani di edifici, terminare il turno.

- ### 3.1.2 Interfaccia hardware

…

- ### 3.1.3 Interfaccia software

…

- ### 3.1.4 Interfaccia di comunicazione

…

3.2 Richieste funzionali

3.2.1 Funzioni del giocatore

3.1.1 Funzione di inizio partita

Permette al giocatore di iniziare una nuova partita tramite il tasto “New Game”.

3.1.2 Funzione di chiusura del gioco

Permette al giocatore di chiudere il gioco tramite il tasto “Exit”.

3.1.3 Funzione di inserimento giocatori

Permette al giocatore di inserire il nome dei giocatori ed iniziare la partita con il tasto “Start Game”.

3.1.4 Funzione di posizionamento worker

Permette al giocatore,nel proprio turno, di posizionare sulla board di gioco i propri worker: prima si seleziona un worker in alto a destra, poi si clicca sulla casella dove si desidera posizionare il worker.

3.1.5 Funzione di fine turno

Permette al giocatore di finire il proprio turno.

3.1.6 Funzione di spostamento worker

Permette al giocatore di spostare,nel proprio turno, un worker dalla casella in cui si trova, ad una casella ad essa adiacente.

3.1.7 Funzione di costruzione delle torri

Permette al giocatore di costruire: La base di una torre se si parte dal livello della board. Un livello di una torre se si ha già una torre, adiacente al worker, di livello inferiore a 3. Una cupola se si ha una torre di livello 3.

3.3 Requisiti di prestazione

Nessun particolare requisito in quanto il software dovrebbe essere abbastanza leggero. Si consiglia però fortemente l'esecuzione su un computer o un laptop.

3.4 Vincoli di progettazione

…

3.5 Attributi del sistema software

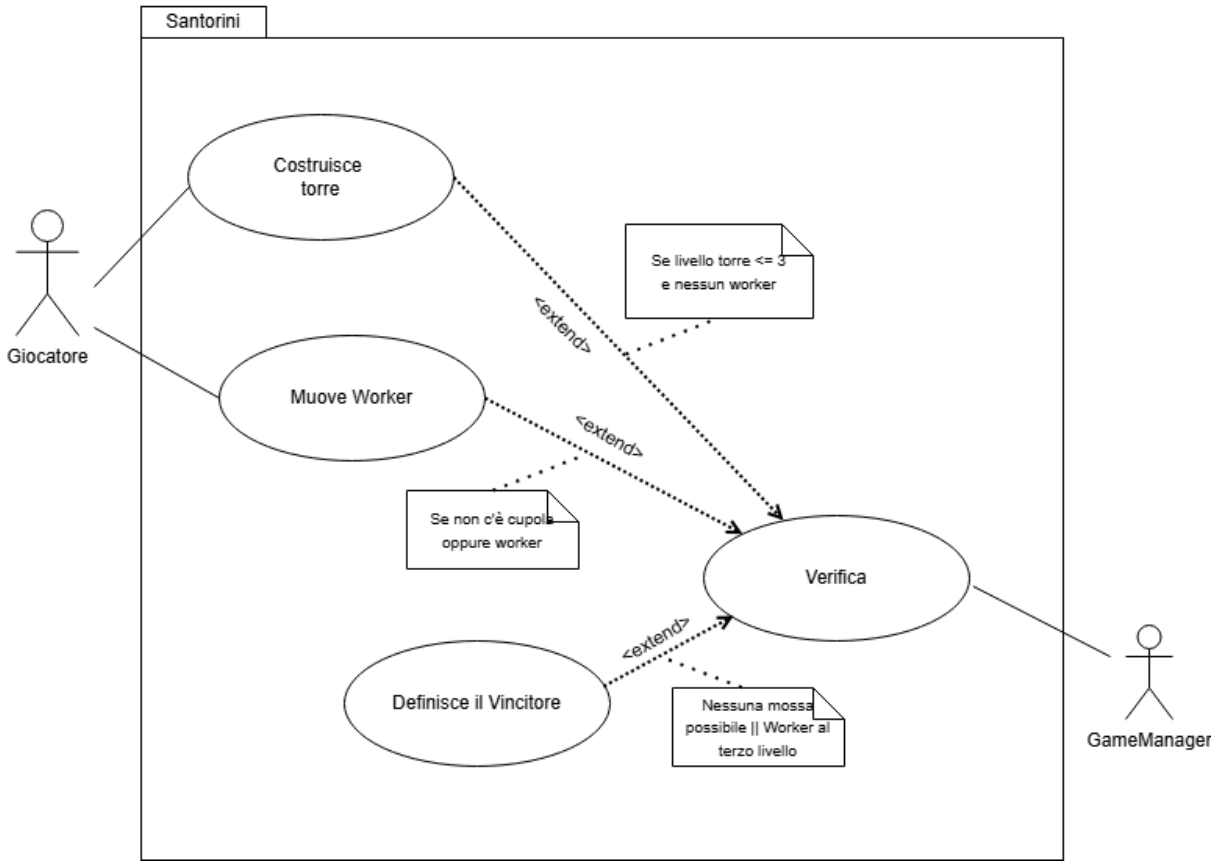
Disponibilità: l'applicazione deve essere disponibile ogni volta che un giocatore vuole usufruirne.

3.6 Altri requisiti

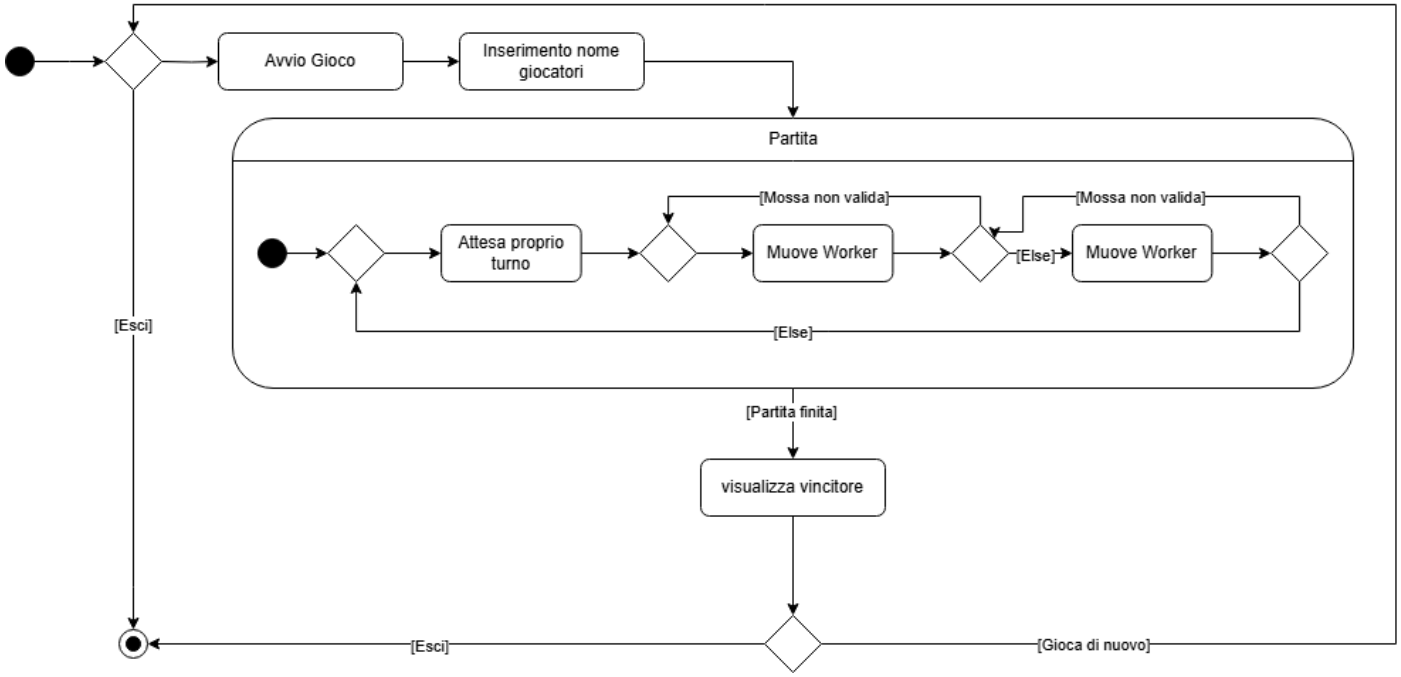
…

Modelling

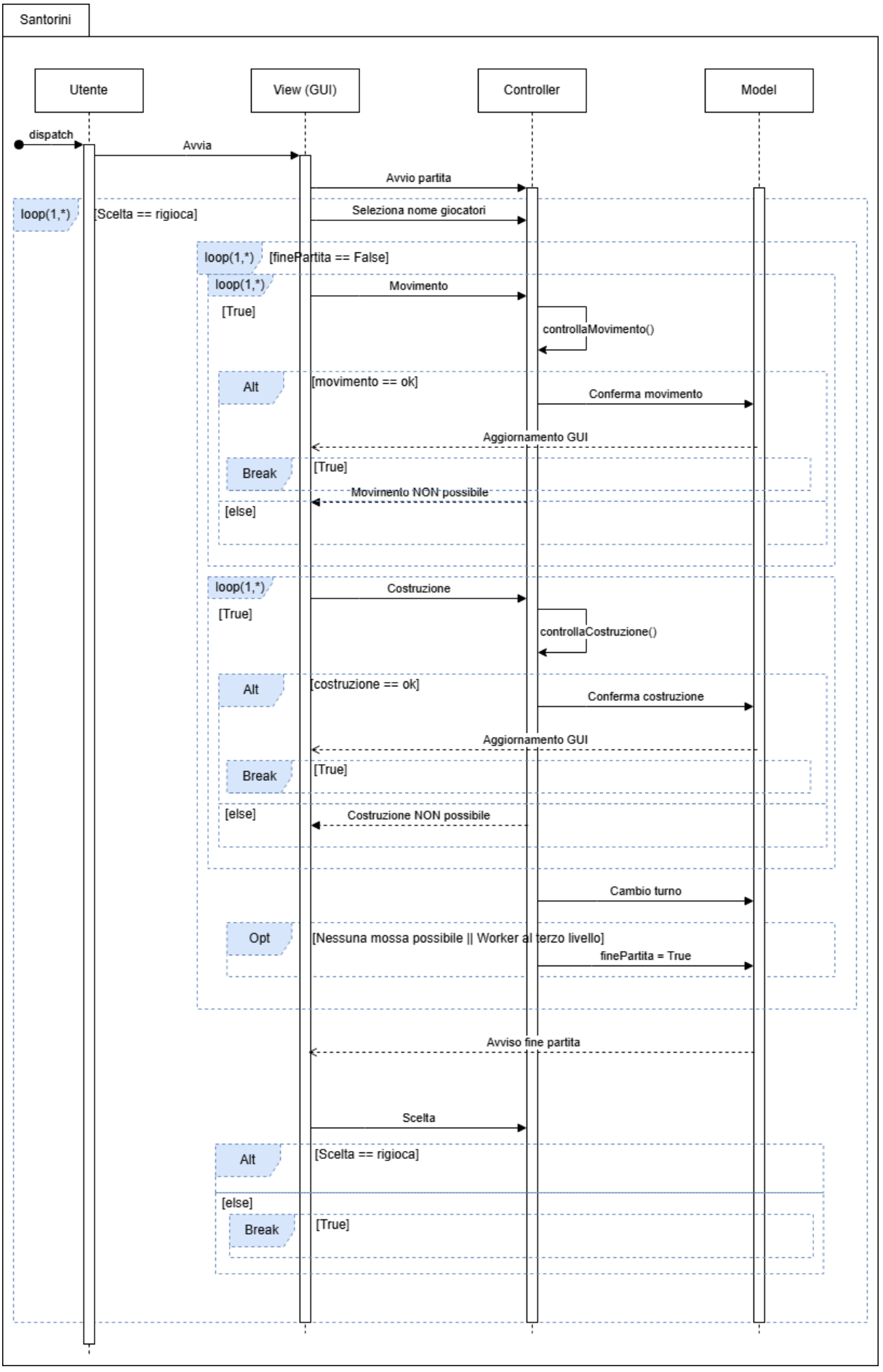
Use case diagram



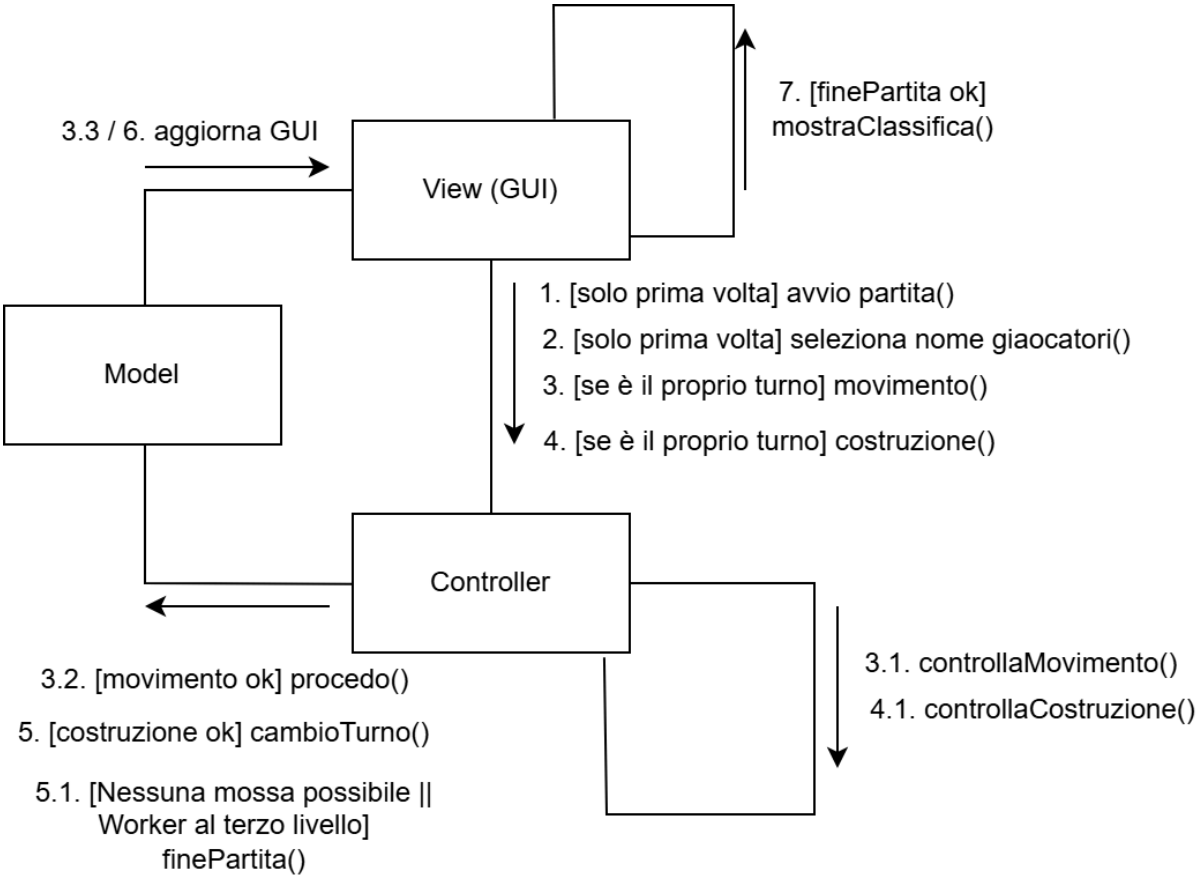
State machine diagram



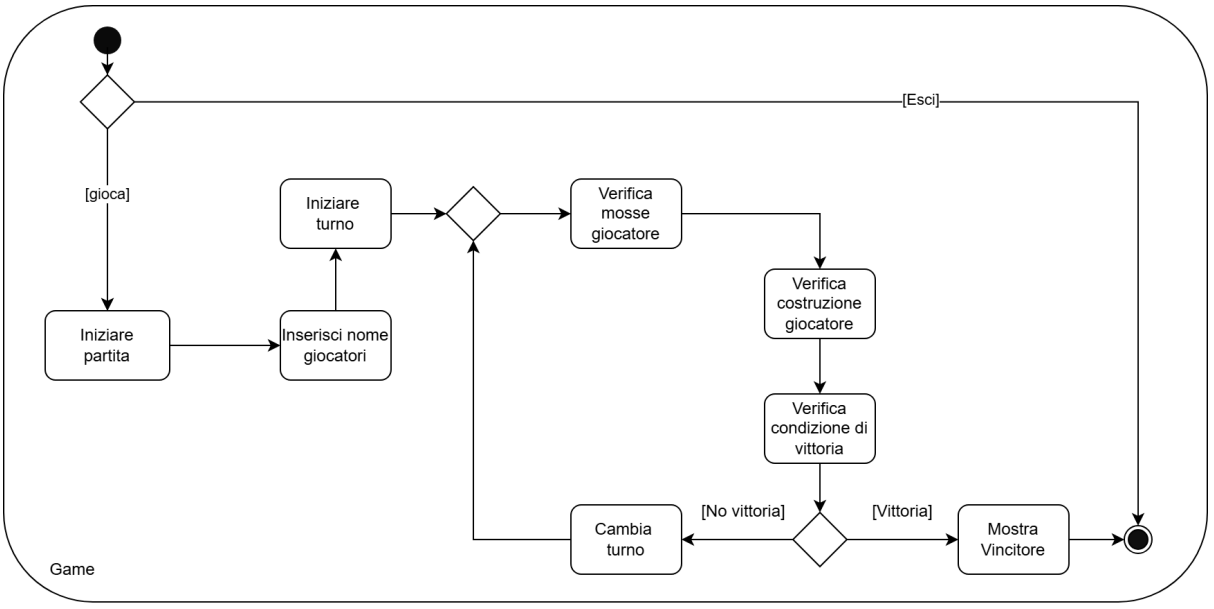
Sequence diagram



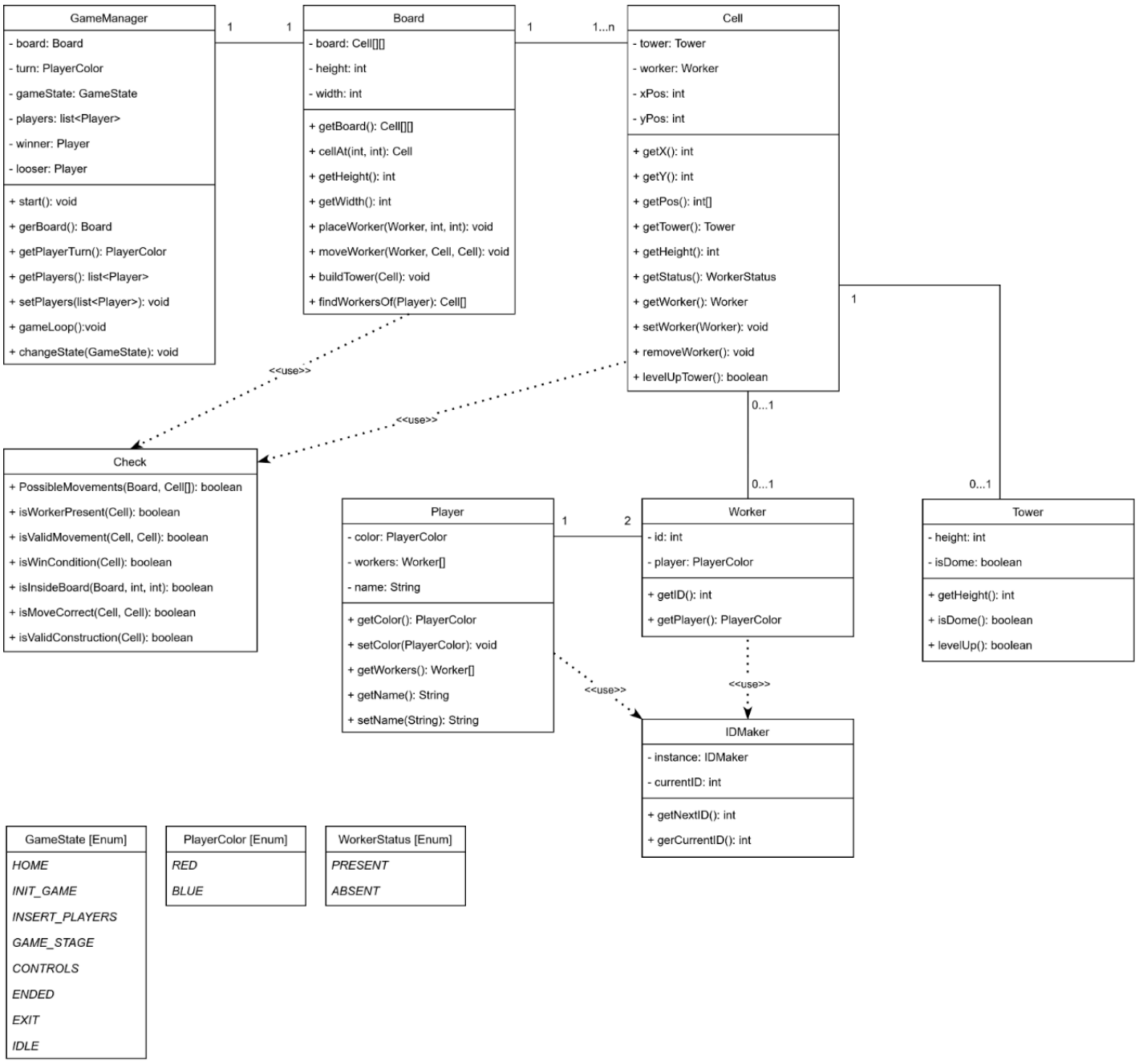
Communication diagram



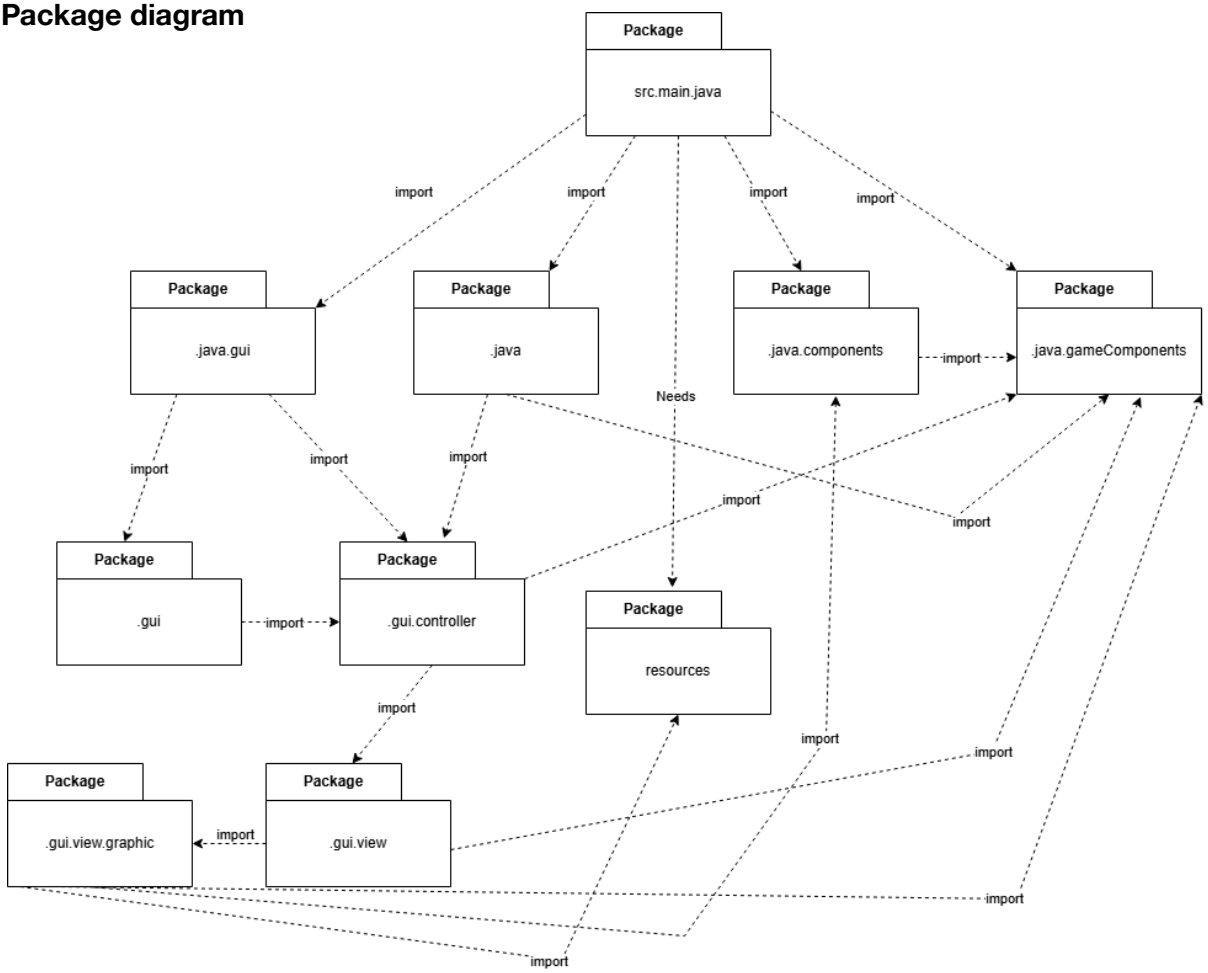
Activity diagram



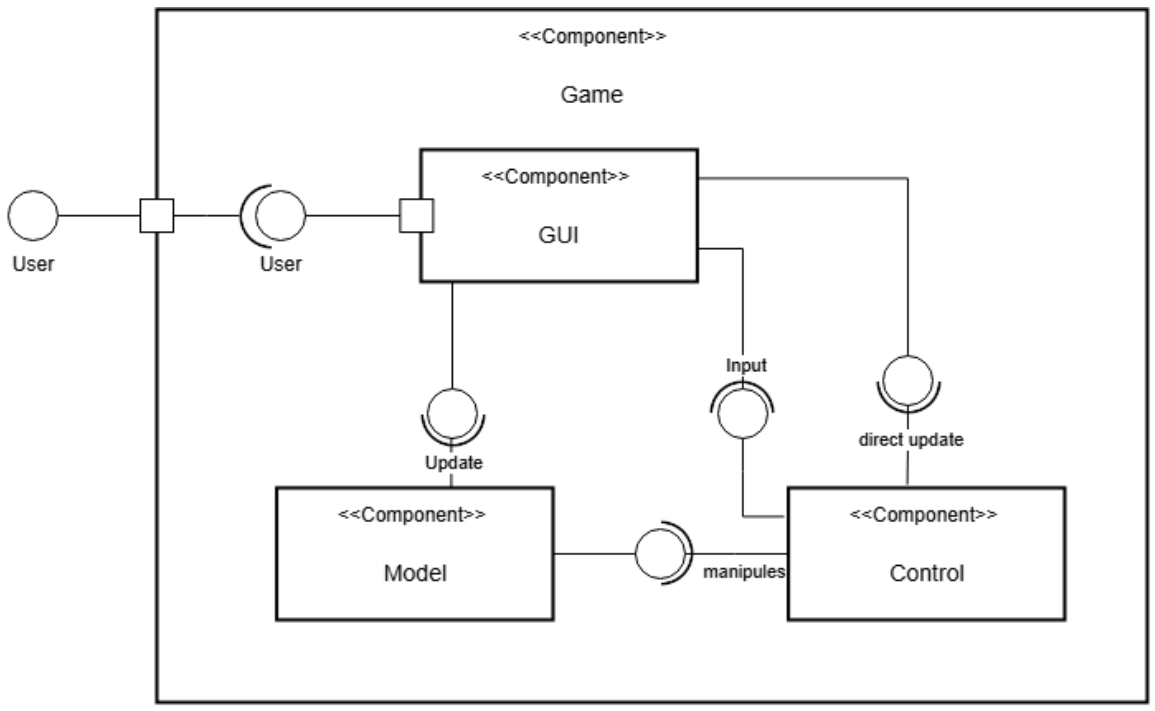
Class diagram



Package diagram



Component diagram



Software Architecture

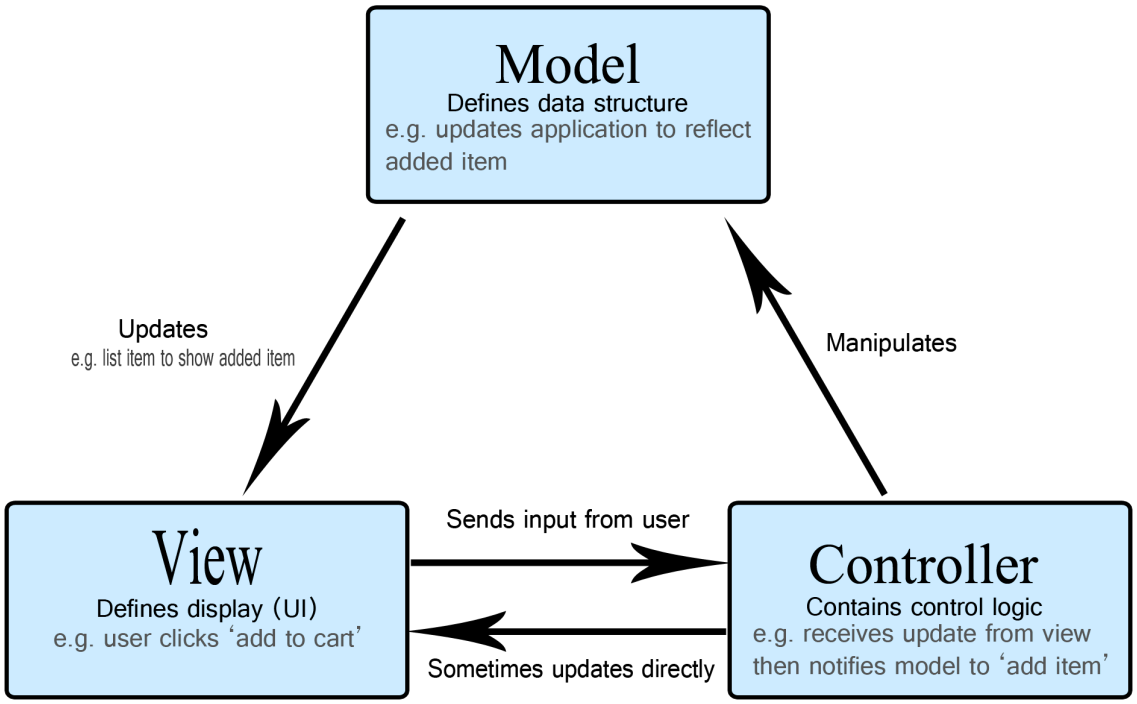
Punti di vista del **modulo (statica)**

- **Classe** : Gli elementi sono descritti come una generalizzazione di altri elementi. La relazione tra gli elementi è 'eredita da'. È ovviamente più applicabile per i sistemi orientati agli oggetti.

Punti di vista dell'**allocazione (realizzativa)**

- **Incarico di lavoro** : Mostra chi sta facendo cosa e aiuta a determinare quale conoscenza è necessaria e dove. Ad esempio, si può decidere di assegnare comunanze funzionali a un unico team.

MVC - stile architetturale

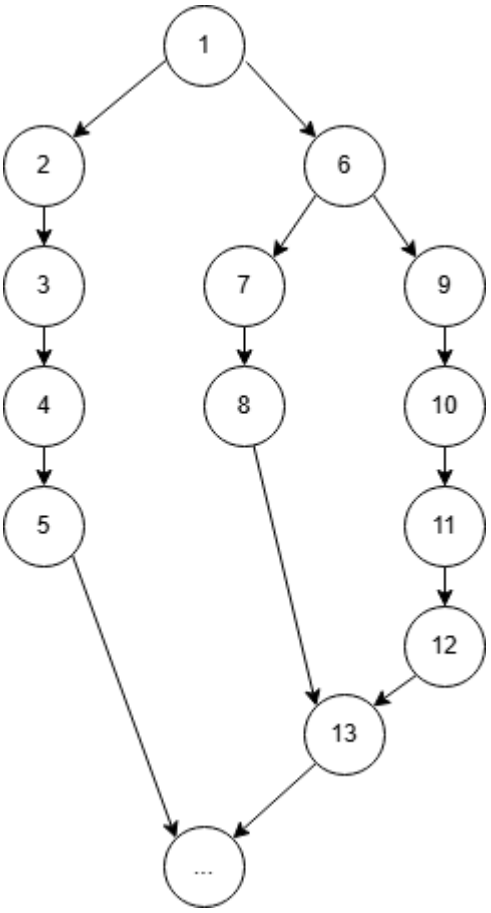


MVC utilizzato per gestire l'interazione tra la GUI e il Backend. Abbiamo utilizzato un GameManager che funge da model, il quale viene modificato dal controller quando nella vista avviene qualche azione (gestita da ActionListener oppure MouseListener), infine il controller aggiorna la View con i nuovi dati del model.

Software Design

McCabe:

```java
/**
 * Updates the player turn and checks if this is the last turn to be done.
 */
public void updatePlayerTurnAndChangeState() {
    if (turn < players.size() - 1) {                //1
        turn++;                                     //2
        turnOpp = 0;                                //3
        changeState(GameState.GAME_STAGE);          //4
    } else {                                        //5
        if (isLastTurn) {                           //6
            changeState(GameState.ENDED);           //7
        } else {                                    //8
            turnOpp++;                              //9
            turn = 0;                               //10
            changeState(GameState.GAME_STAGE);      //11
        }                                           //12
    }                                               //13
}
```



CV = e - n + p + 1 = 13 - 13 + 1 + 1 = 2

Codice misurato con Sonarqube, eventuali errori rilevati anche con PMD.

Viene utilizzato il singleton pattern.

Software Testing

I test sono stati eseguiti su tutte i metodi delle classi del pacchetto “gameComponents”.
Ecco alcuni esempi di testing:

- Board, movimento degli worker

```
/*
 * This test aims to check the moveWorker method functionality
 */
@Test
void testMoveWorker() {
    Player player = new Player("A", PlayerColor.RED);
    Worker worker = player.getWorkers()[0];
    board.placeWorker(worker, 1, 1);
    Cell startingCell = board.cellAt(1, 1);
    Cell finalCell = board.cellAt(2, 2);
    board.moveWorker(worker, startingCell, finalCell);
    assertNull(startingCell.getWorker(), "Starting cell should have no worker after move");
    assertEquals(worker, finalCell.getWorker(), "Worker should be moved to final cell");
}
```

- Cell, aumento altezza torre

```
/*
 * This test aims to check the levelUpTower method functionality
 */
@Test
void testLevelUpTower() {
    Tower tower = cell.getTower();
    assertTrue(cell.levelUpTower(), "First level up should be successful");
    assertEquals(1, tower.getHeight(), "Tower height should be 1 after first level up");
    assertFalse(tower.isDome(), "Tower should not be a dome at height 1");

    // Level up to dome
    cell.levelUpTower(); // height 2
    cell.levelUpTower(); // height 3
    assertTrue(cell.levelUpTower(), "Level up to dome should be successful");
    assertEquals(4, tower.getHeight(), "Tower height should be 4");
    assertTrue(tower.isDome(), "Tower should be a dome at height 4");

    // Attempt to level up beyond dome
    assertFalse(cell.levelUpTower(), "Level up should fail when tower is already a dome");
    assertEquals(4, tower.getHeight(), "Tower height should remain at 4");
}
```

- Check , controllo corretto movimento

```
/*
 * This test aims to check the isMoveCorrect method functionality
 */
@Test
void testIsMoveCorrect() {
    Cell start = board.cellAt(2, 2);
    Cell same = board.cellAt(2, 2);
    Cell adjacent = board.cellAt(2, 3);
    Cell diagonal = board.cellAt(3, 3);
    Cell far = board.cellAt(4, 4);

    assertFalse(Check.isMoveCorrect(start, same), "Moving to the same cell should be incorrect");
    assertTrue(Check.isMoveCorrect(start, adjacent), "Moving to an adjacent cell should be correct");
    assertTrue(Check.isMoveCorrect(start, diagonal), "Moving to a diagonally adjacent cell should be correct");
    assertFalse(Check.isMoveCorrect(start, far), "Moving more than one cell away should be incorrect");
}
```

Software Maintenance

Sono state svolte due attività di manutenzione:

- **Manutenzione correttiva**: la versione sviluppata conteneva parecchi bugs nella parte del GameStage (bugs relativi alla parte grafica).
- **Manutenzione preventiva**: miglioramento di alcuni metodi per renderli più comprensibili/usabili e quindi più manutenibili.

In questa fase dello sviluppo la manutenzione può essere un problema in quanto si può incorrere in qualche parte di codice non strutturato, oppure si può avere una conoscenza del dominio insufficiente (riguardanti le tecnologie da usare nel progetto stesso oppure non conoscenza della complessità di future parti del progetto) e la documentazione potrebbe risultare insufficiente per via del parallelismo con la stesura del codice.

Le principali **attività di refactoring** svolte sono:

- ➔ Cambiare la dichiarazione di alcune funzioni per rendere il **codice più comprensibile**(nomi significativi che spiegano il comportamento effettivo delle funzioni)
- ➔ **Parametrizzare una funzione**, molto utilizzata durante la fase di sviluppo dell'interfaccia grafica (GUI) in quando per la stampa dei vari oggetti si ha bisogno delle coordinate dettate dai pixel, perciò si è fatto un refactoring per ottenere un parametro nella funzione nella quale viene variato (es. moltiplicato) ogni qualvolta che viene richiamata la funzione.
- ➔ **Rimozione codice non utilizzato**, questo ha fatto sì che alcune classi si alleggerissero a livello di complessità e in termini di linee di codice.
- ➔ **Rimozione dei MiddleMan**, per il ricavo delle informazioni relative ad un oggetto ci si riferisce direttamente alla classe nella quale è contenuto.
- ➔ **Rimozione di (alcuni) metodi di set**, in delle classi i metodi di set non erano utili.
- ➔ Rimosso alcuni campi di alcune classi per mancanza di utilizzo.
- ➔ **Ridenominazione di variabili**, in modo tale che i nomi delle variabili si adattino meglio alla loro funzioni.
- ➔ **Rimozione di (alcuni) controlli** utilizzando "break" oppure "continue". Attività eseguita per snellire e ridurre il numero di righe di codice che potrebbero portare, con l'accumulo di complessità, a un calo di prestazioni.
- ➔ **Rimozione di (alcuni) errori di codice** tramite delle Exception, in particolare con la creazione di nuove Exception per gestire la mancanza di un model oppure di una view nella struttura MVC.

➔ **Sostituzione di (alcuni) algoritmi** per il miglioramento delle prestazioni di metodi.