Università degli Studi di Napoli “Parthenope” Facoltà di Scienze e Tecnologie
Corso di laurea in Informatica (percorso Generale)


Relazione Progetto - Programmazione III

# Connect 4 - Forza 4

Docente
Angelo Ciaramella

Candidato
Ciro Cozzolino
0124001804

Anno Accademico 2019-2020

# Introduzione

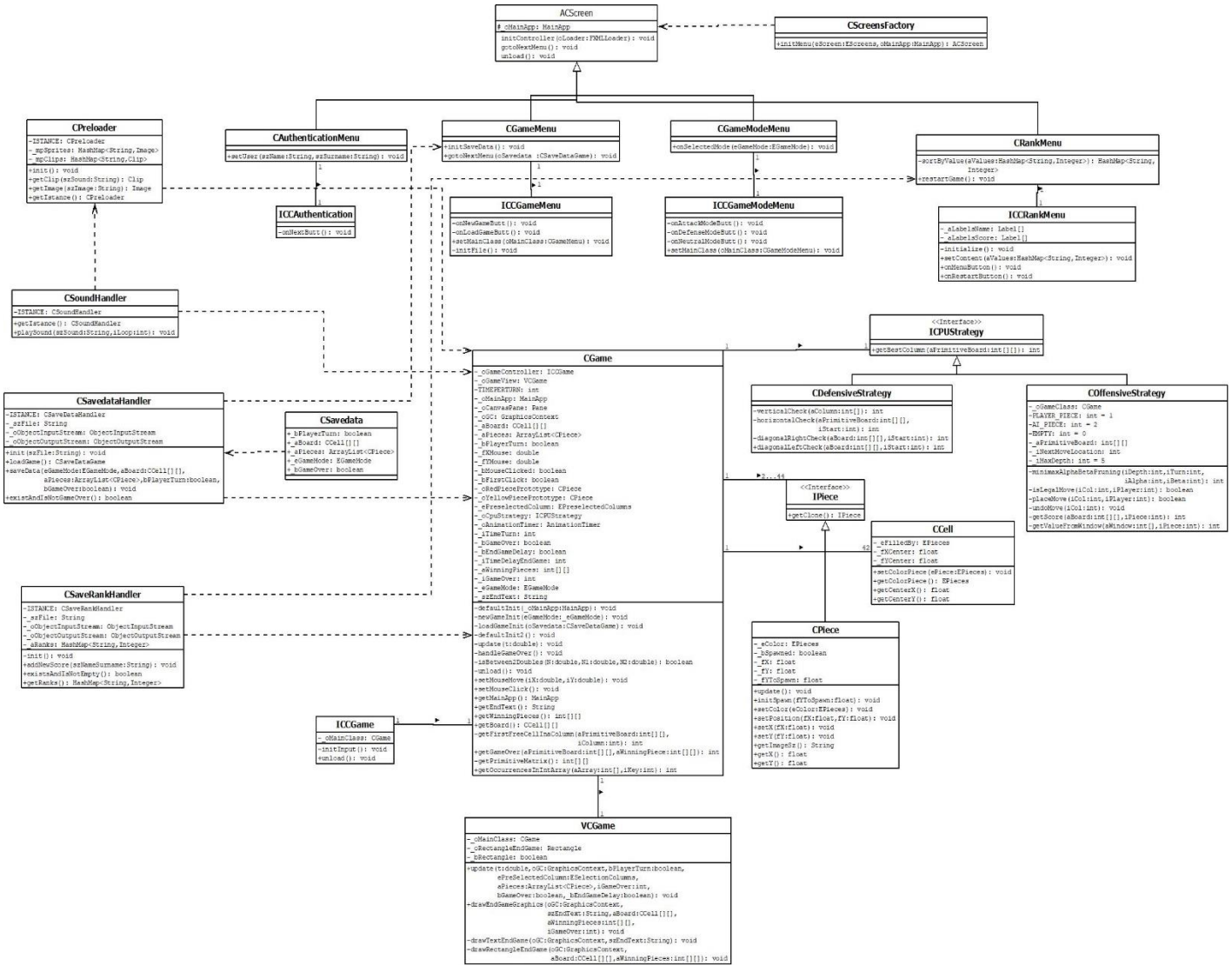
Per questo esame va esposto un progetto, nel mio caso un videogioco: Forza 4
Forza 4 è un gioco da tavolo astratto di allineamento prodotto dalla Milton Bradley nel 1974.
Si tratta di un gioco di allineamento (di solito su una matrice di 6 righe e 7 colonne) che richiama i giochi del Filetto e del Tris.
Anche in questo caso l'obiettivo è mettere in fila (orizzontale, verticale o diagonale) un certo numero (quattro nel gioco base, e da qui il suo nome) di proprie pedine; ma l'elemento fondamentale del gioco, che lo rende del tutto originale, è la forza di gravità: la scacchiera è posta in verticale fra i giocatori, e le pedine vengono fatte cadere lungo una griglia verticale, in modo tale che una pedina inserita in una certa colonna va sempre a occupare la posizione libera situata più in basso nella colonna stessa.

# Requisiti

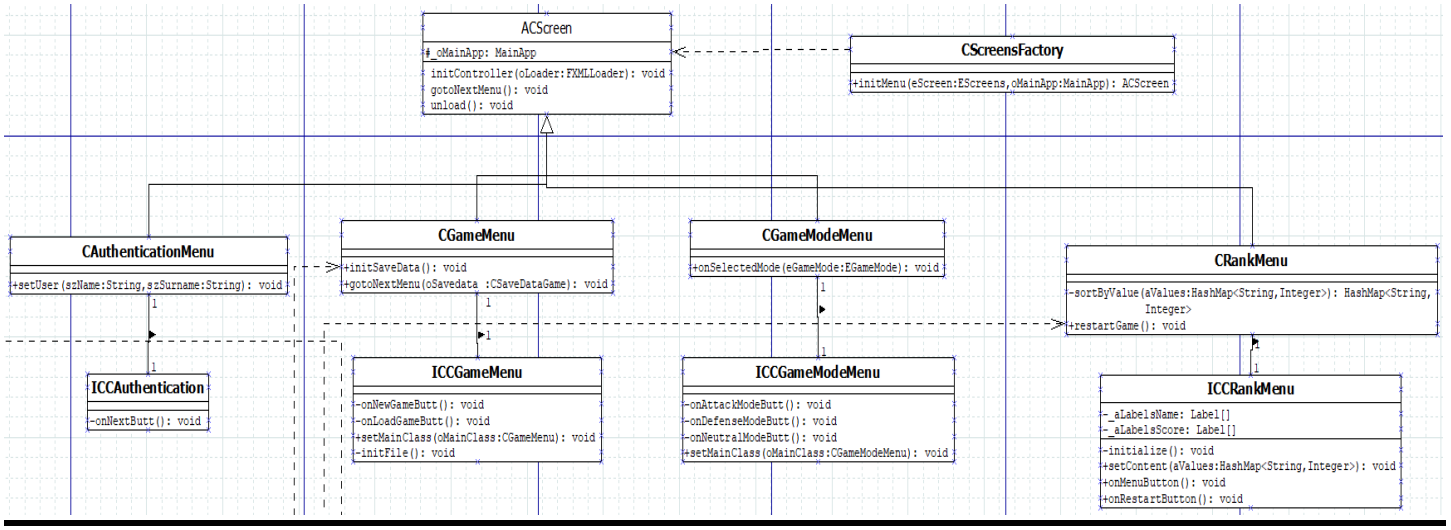
I requisiti per realizzare questo progetto sono stati:

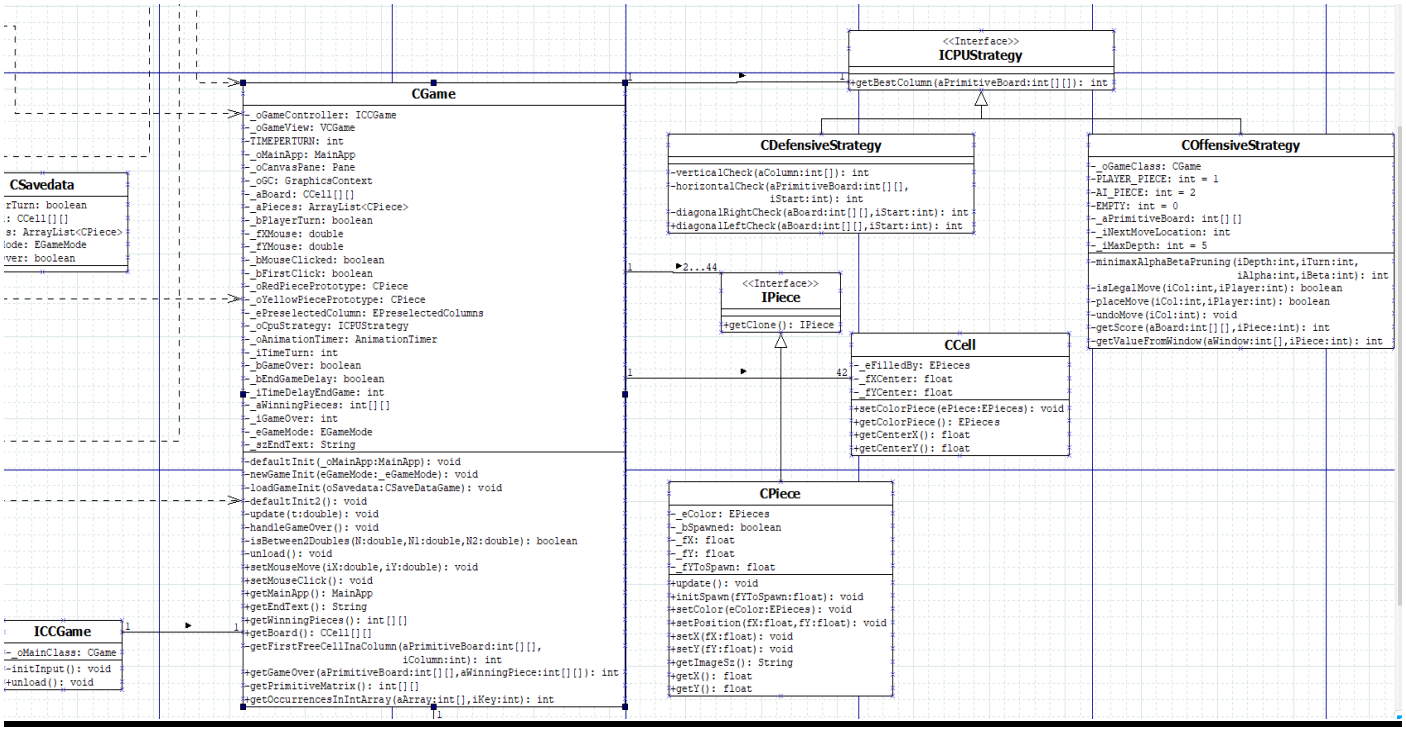
- Una buona conoscenza del gioco forza 4 (Regole)
- Una buona conoscenza delle strategie migliori del gioco
- Una buona conoscenza del funzionamento dell'algoritmo dell'albero Mini-max con alpha-beta pruning (Potatura)
- Una buona conoscenza del linguaggio Java e della programmazione Object-Oriented
- Design Pattern Strategy per applicare le diverse strategie della CPU
- Design Pattern Prototype per la creazione delle pedine
- Design Pattern Factory Method per la creazione delle schermate non in game

# Diagramma delle classi

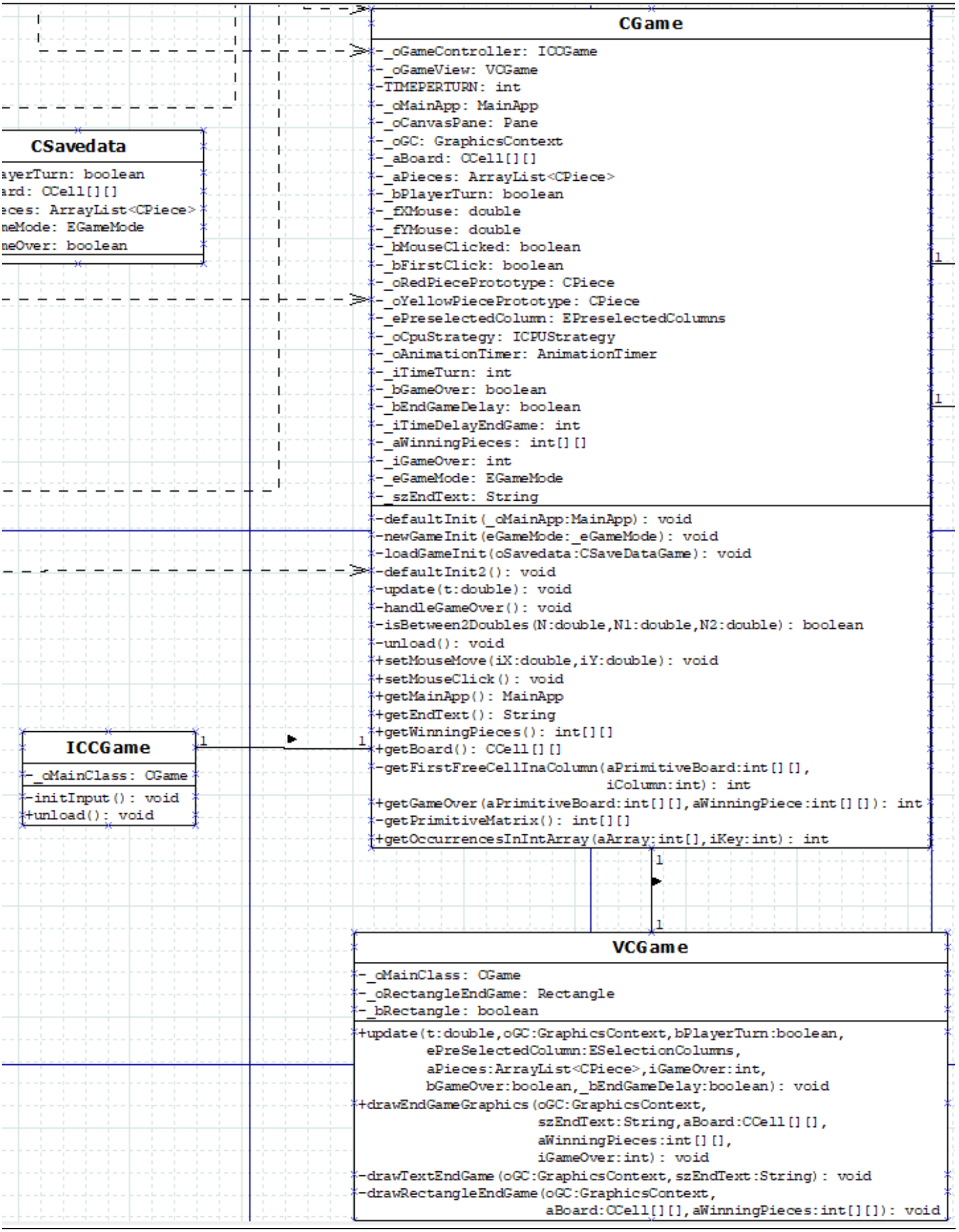


# Diagramma delle classi

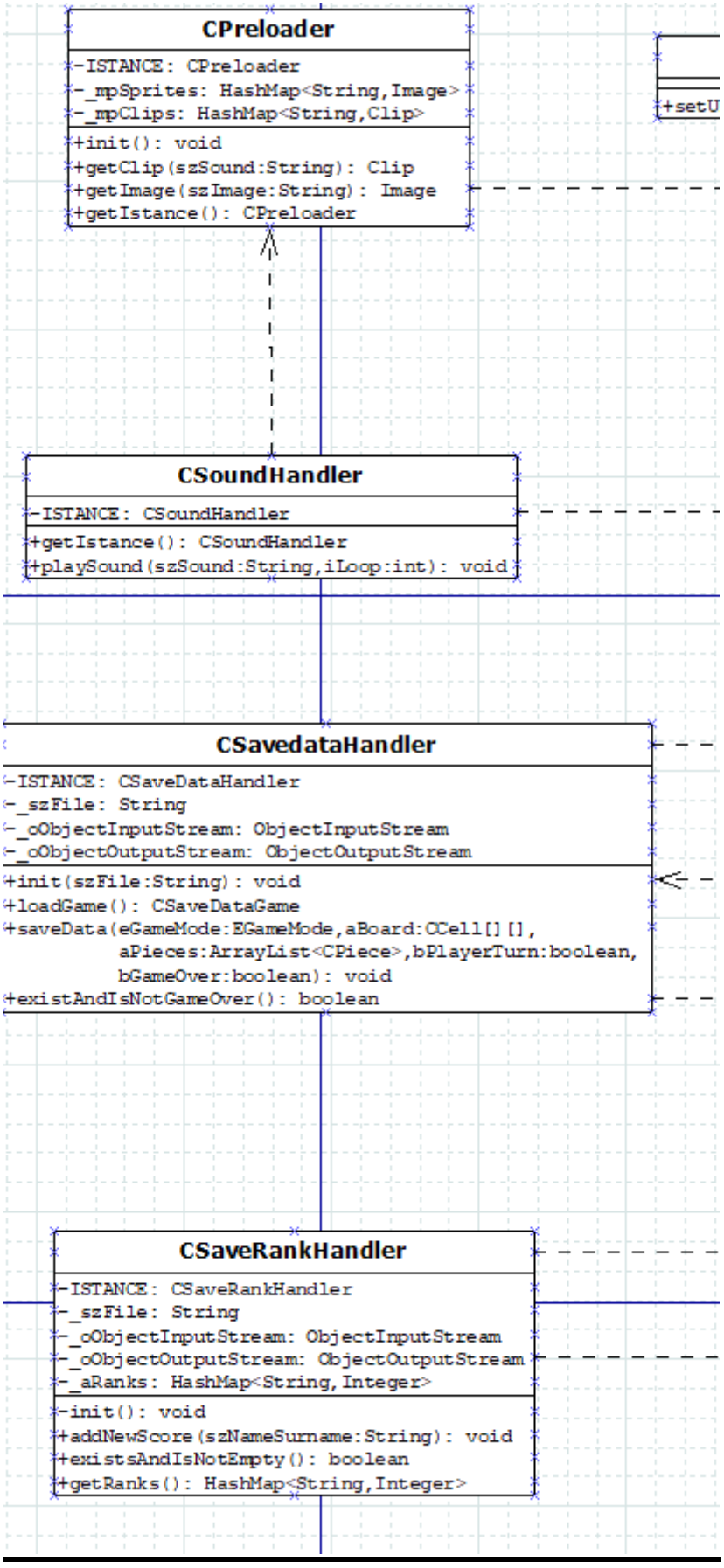


# Diagramma delle classi



# Diagramma delle classi



# Parti rilevanti del codice

## -Minimax con alpha beta pruning

```
private int minimaxAlphaBetaPruning(int iDepth, int iTurn, int iAlpha, int iBeta)
{
    //Casi Base

    if (iBeta <= iAlpha) //Se beta è maggiore di alpha vuol dire che non ha senso continuare a cercare attraverso quest albero
    {
        if (iTurn == AI_PIECE)
            return Integer.MAX_VALUE; //Ritorna il valore massimo se il turno è della cpu
        else
            return Integer.MIN_VALUE; //Ritorna il valore minimo se il turno è del player
    }

    int iGameOver = _oGameClass.getGameOver(_aPrimitiveBoard,null);
    if (iGameOver != 0) //Controllo se c'è una situazione di gameover
    {
        if (iGameOver == AI_PIECE) //In caso di vittoria della cpu, ritorna il valore massimo / 2
            return Integer.MAX_VALUE/2;
        else if (iGameOver == PLAYER_PIECE) //In caso di vittoria del player, ritorna il valore minimo / 2
            return Integer.MIN_VALUE/2;
        else
            return 0; //In caso di pareggio, ritorna 0
    }

    if (iDepth == 0) //Se la profondità massima è stata raggiunta, ritorna il valore secono la funzione di valore euristico
        return getScore(_aPrimitiveBoard, AI_PIECE);

    int iMaxScore = Integer.MIN_VALUE, iMinScore = Integer.MAX_VALUE; //Inizializza MaxScore al minimo e MinScore al massimo

    for (int i=0;i<6;i++) //Itera tutte le colonne della scacchiera
    {
        int iCurrentScore = 0; //Inizializza score a 0

        if (!isLegalMove(i)) continue; //se la mossa per questa colonna non è valida, passa alla colonna successiva

        if (iTurn == AI_PIECE) //Se è il turno della cpu, quindi massimizza
        {
            placeMove(i, AI_PIECE); //Inserisce la pedina della cpu nella colonna iterata
            iCurrentScore = minimaxAlphaBetaPruning(iDepth-1, PLAYER_PIECE, iAlpha, iBeta); // chiamata ricorsiva con profondità minore di 1 e secondo il turno del player

            if (iDepth == _iMaxDepth) //Se ci troviamo a profondità = iMaxDepth vuol dire che siamo risaliti al primo nodo e quindi:
            {
                if (iCurrentScore > iMaxScore) _iNextMoveLocation = i; //Vedo se lo score di questo nodo è maggiore del valore massimo, e in tal caso salvo globalmente
                                                                       //l'indice della colonna migliore
                if (iCurrentScore == Integer.MAX_VALUE/2) //Se lo score attuale è lo score di vittoria
                {
                    undoMove(i); //Rimuovo la pedina nella colonna i;
                    break; //interrompo il ciclo;
                }
            }

            iMaxScore = Math.max(iCurrentScore, iMaxScore);
            iAlpha = Math.max(iCurrentScore, iAlpha);
```

```
        }else if(iTurn == PLAYER_PIECE) //Se è il turno del player, quindi minimizza
        {
            placeMove(i, PLAYER_PIECE); //Inserisco la pedina del player nella colonna iterata
            iCurrentScore = minimaxAlphaBetaPruning(iDepth-1, AI_PIECE, iAlpha, iBeta); //Chiamata ricorsiva con profondità minore di 1 secondo il turno della cpu
            iMinScore = Math.min(iCurrentScore, iMinScore);
            iBeta = Math.min(iCurrentScore, iBeta);
        }

        undoMove(i); //Rimuovo la pedina nella colonna i;
        if (iCurrentScore == Integer.MAX_VALUE || iCurrentScore == Integer.MIN_VALUE) break;
    }
    return iTurn==AI_PIECE?iMaxScore:iMinScore; //Se è il turno della cpu ritorna max score, se è quello del player ritorna min score

}
```

# Parti rilevanti del codice

## Funzione di salva partita

```
public void saveData(EGameMode eGameMode, CCell[][] aBoard,ArrayList<CPiece> aPieces, boolean bPlayerTurn, boolean bGameOver) //Funzione per salvare lo stato della partita in un file
{
    new File(_szFile);
    CSaveDataGame oSavedata = new CSaveDataGame(); //crea l'oggetto
    //Lo riempio
    oSavedata._eGameMode = eGameMode;
    oSavedata._aBoard = aBoard;
    oSavedata._aPieces = aPieces;
    oSavedata._bPlayerTurn = bPlayerTurn;
    oSavedata._bGameOver = bGameOver;

    try
    {
        _oObjectOutputStream = new ObjectOutputStream(new FileOutputStream(_szFile));
        _oObjectOutputStream.writeObject(oSavedata); //Lo Scrivo sul file
        _oObjectOutputStream.close();
    }catch(IOException e)
    {
        e.printStackTrace();
    }

}
```

## Funzione di carica partita

```
public CSaveDataGame loadGame() //Funzione di carica partita
{
    CSaveDataGame oSavedata = new CSaveDataGame();
    try
    {
        _oObjectInputStream = new ObjectInputStream(new FileInputStream(_szFile));
        oSavedata = (CSaveDataGame)_oObjectInputStream.readObject(); //Legge l oggetto dal file
        _oObjectInputStream.close();
    }catch(IOException e)
    {
        e.printStackTrace();
    }catch(ClassNotFoundException e)
    {
        e.printStackTrace();
    }

    return oSavedata; //Ritorna l oggetto letto
}
```

# Parti rilevanti del codice

## Classe astratta per il Factory Method

```java
package application;

import java.io.IOException;

/**
 * <p>Classe Astratta per le interfacce grafiche (Non in game)</p>
 * @author Ciro Cozzolino
 */

public abstract class ACScreen { //Classe astratta per le interfacce grafiche non in game.

    protected MainApp _oMainApp; //Riferimento al main

    /**
     *
     * @param oMainApp Riferimento al main.
     * @param szLoaderPath Path del file fxml.
     */

    public ACScreen(MainApp oMainApp, String szLoaderPath)
    {
        _oMainApp = oMainApp;
        try {
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(MainApp.class.getResource(szLoaderPath)); //Cerca la schermata nel file fxml
            AnchorPane oMenu = (AnchorPane) loader.load(); //La carica
            _oMainApp.getRootLayout().setCenter(oMenu); //La seleziona come pannello centrale del Root Layout
            this.initController(loader); //funzione astratta per inizializzare la classe input controller della stessa;

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    protected abstract void initController(FXMLLoader oLoader); //funzione per inizializzare la classe controller

    public abstract void gotoNextMenu(); //funzione astratta per passare alla prossima schermata

    protected abstract void unload(); //funzione astratta per la chiusura della schermata

}
```

## Classe creator per il Factory Method

```java
package application;

/**
 * <p>Classe per il pattern Factory Method delle schermate (Non in gioco)</p>
 * @author Ciro Cozzolino
 */

public class CScreensFactory {
    public CScreensFactory() {}

    /**
     * <p>Dato come parametro, la schermata da inizializzare, instanzia una delle classi figlie di ACScreen (Schermate)</p>
     * @param eScreen Enum che rappresenta la schermata da inizializzare
     * @param oMainApp Riferimento al main
     * @return Figlio di ACScreen o null
     */

    public ACScreen initMenu(EScreens eScreen, MainApp oMainApp) //Classe Factory Method per l'inizializzazione delle schermate
    {
        switch (eScreen)
        {
            case AUTHENTICATION_MENU:
                return new CAuthenticationMenu(oMainApp);
            case GAME_MENU:
                return new CGameMenu(oMainApp);
            case GAME_MODE_MENU:
                return new CGameModeMenu(oMainApp);
            case RANKSMENU:
                return new CRankMenu(oMainApp);
            default:
                return null;
        }
    }
}
```

# Parti rilevanti del codice

### Interfaccia per il design pattern Strategy

```java
package application;

/**
 * <p>Interfaccia per lo Strategy Pattern</p>
 * @author Ciro Cozzolino
 */

public interface ICPUStrategy { //Interfaccia per lo Strategy Pattern
    /**
     * <p>Funzione per ottenere la migliore mossa secondo l'attuale strategia</p>
     * @param aPrimitiveBoard Matrice primitiva della tavola da gioco
     * @return Int (colonna migliore)
     */
    public int getBestColumn(int[][] aPrimitiveBoard);
}

```

### Interfaccia per il design Pattern Prototype

```java
package application;

/**
 * <p>Interfaccia per l'utilizzo del Prototype Pattern per le pedine</p>
 * @author Ciro Cozzolino
 */

public interface IPiece extends Cloneable{ //Interfaccia per il prototype pattern
    /**
     * @return un nuovo elemento IPiece, uguale a quest'ultimo ma con un area di memoria differente
     */
    public IPiece getClone();
}

```