Escuela de Computación - San José

MEDIEVAL'S

Informe de Proyecto

José Ceciliano Granados
Silvia Calderón Navarro

Vercíon 1.0
8 de abril de 2017

Enunciado

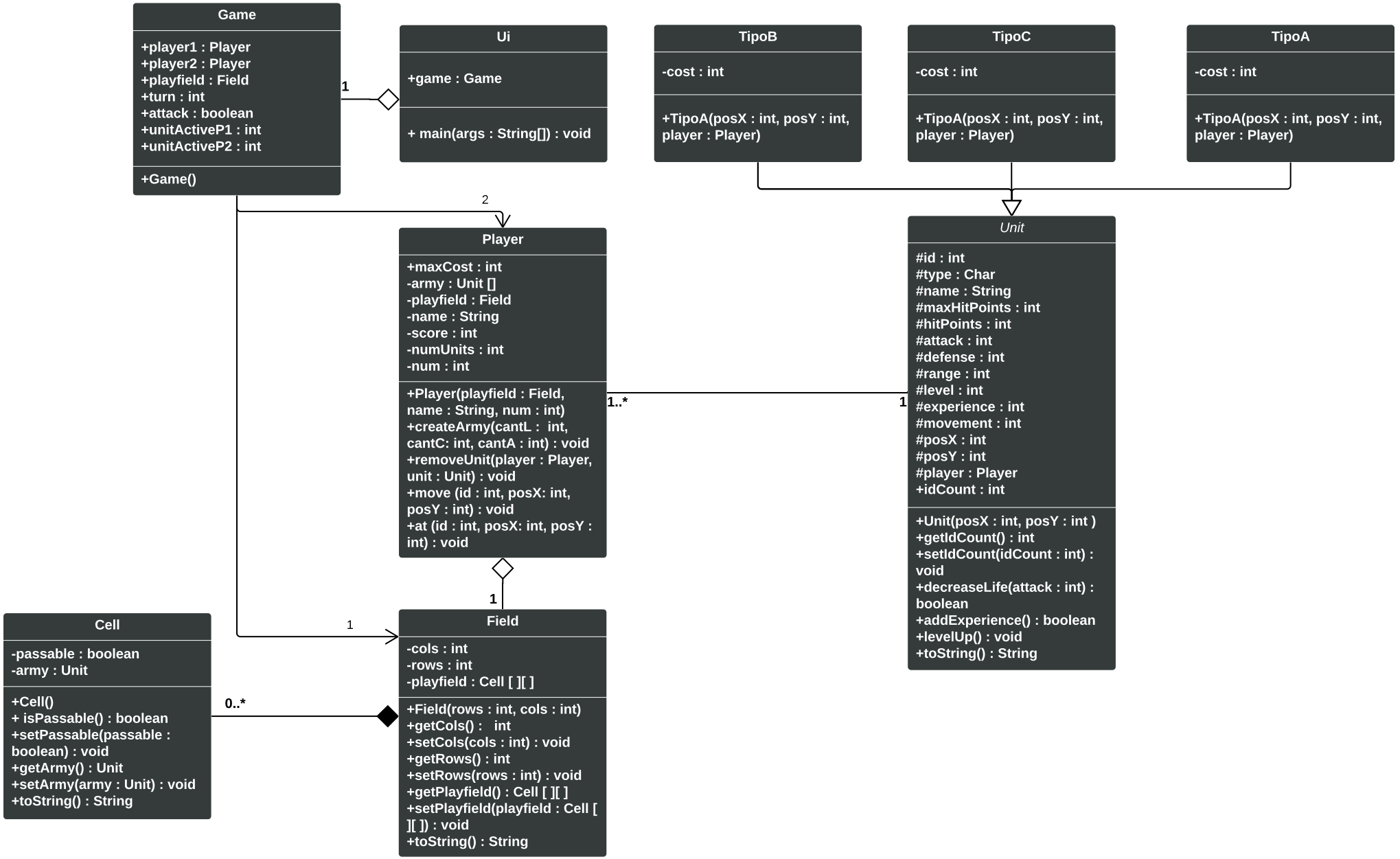
El objetivo de este proyecto es brindarle al estudiante la capacidad de modelar e implementar problemas utilizando la programación orientada a objetos usando el lenguaje de programación Java. Se debe programar y documentar un motor básico para un juego de estrategia por turnos en modo texto. Algunos ejemplos de estos juegos son Final Fantasy Tactics o Fire Emblem. En este tipo de juegos, unidades de bandos opuestos se enfrentan en un campo de juego, hasta que uno de los bandos quede sin unidades. Las unidades reciben da˜no basado en sus habilidades de defensa y ofensiva del rival hasta que sus puntos de vida se acaben. Cada vez que una unidad derrota a un rival, esta gana experiencia y cuando acumula la suficiente, sube un nivel, restaurando sus puntos de vida y mejorando sus habilidades.

Dentro de los principales puntos básicos del juego tendrá:

- Un sistema de apertura en el cual se podrá elegir iniciar un nuevo juego o resturar un juego anterior.
- Tener un apartado que permita elegir el nombre y ejercito de cada uno de los jugadores además de permitir ver la cantidad de monedas restantes del jugador correspondiente.
- Un tablero que muestre el turno del jugador así como la posibilidad de mover o atacar según lo determine el jugador, además de una tabla con la ficha actual y sus valores correspondientes como lo son la defensiva y ofensiva entre otros.
- La posibilidad de guardar un juego con todos los estados actuales y permitir sobrescribir juegos anteriormente guardados.
- La interfaz debe permitir salir en todo momento para facilidad del usuario y validar los datos del usuario al ser ingresados.

Implementación

- Dentro de la carpeta Dist se encuentra la documentación del JavaDoc mas especifico con cada implementación de los métodos
- Dentro de una clase game tenemos dos jugadores y un campo de juego.
- Dentro del campo de juego contenemos un playfiel la cual mantiene las fichas colocadas en el campo de juego y mantiene un arreglo de celdas del juego como tal.
- Los Jugadores tienen un nombre y un arreglo de jugadores que a su vez tienen atributos los cuales son definidos por la tabla impuesta como lineamiento para este programa.
- Cada movimiento coloca a un jugador en una posición X y Y designada con respecto al los demás jugadores.
- Cada ataque realiza una serie de cálculos para disponer de los atributos del otro jugador los cuales son impuestos de acuerdo a lo impuesto por el programa.
- Los juegos son guardados con métodos de serialize por lo cual todas las clases tienen el implements serialize, para guardar sus objetos y luego reconstruirlos.
- La lógica de los archivos es que se guarda un archivo nombre 1.obj para el primer juego realizado y así sucesivamente.
- Los objetos guardados en los archivos son escritos y llamados por su tipo, en los cuales tenemos los jugadores,el playfuel, y el modo si es de ataque o no, si es un movimiento o no, y el turno del jugador, además de todas las posiciones de las fichas en el momento.


Game
+player1 : Player
+player2 : Player
+playfield : Field
+turn : int
+attack : boolean
+unitActiveP1 : int
+unitActiveP2 : int
+Game()
1
Ui
+game : Game
+ main(args : String[]) : void
TipoB
-cost : int
+TipoA(posX : int, posY : int, player : Player)
TipoC
-cost : int
+TipoA(posX : int, posY : int, player : Player)
TipoA
-cost : int
+TipoA(posX : int, posY : int, player : Player)
2
Player
+maxCost : int
-army : Unit []
-playfield : Field
-name : String
-score : int
-numUnits : int
-num : int
+Player(playfield : Field, name : String, num : int)
+createArmy(cantL : int, cantC: int, cantA : int) : void
+removeUnit(player : Player, unit : Unit) : void
+move (id : int, posX: int, posY : int) : void
+at (id : int, posX: int, posY : int) : void
1..*
1
Unit
#id : int
#type : Char
#name : String
#maxHitPoints : int
#hitPoints : int
#attack : int
#defense : int
#range : int
#level : int
#experience : int
#movement : int
#posX : int
#posY : int
#player : Player
+idCount : int
+Unit(posX : int, posY : int)
+getIdCount() : int
+setIdCount(idCount : int) : void
+decreaseLife(attack : int) : boolean
+addExperience() : boolean
+levelUp() : void
+toString() : String
1
1
Cell
-passable : boolean
-army : Unit
+Cell()
+ isPassable() : boolean
+setPassable(passable : boolean) : void
+getArmy() : Unit
+setArmy(army : Unit) : void
+toString() : String
0..*
Field
-cols : int
-rows : int
-playfield : Cell [ ][ ]
+Field(rows : int, cols : int)
+getCols() : int
+setCols(cols : int) : void
+getRows() : int
+setRows(rows : int) : void
+getPlayfield() : Cell [ ][ ]
+setPlayfield(playfield : Cell [ ][ ]) : void
+toString() : String

Estructura del Repositorio

A continuación mostraremos la distribución del repositorio en GitLab:

- Codigo:

  Raíz: se encuentran los archivos por defecto de Netbeans y además tenemos tres archivos .obj los cuales guardan los objetos necesarios para reconstruir una partida desde el juego (restaurar juego).

  Build: Construcción básica de las clases impuestas

  Dist: Proyecto1.jar: Construcción jar del proyecto si se tiene instalado una versión de java en las carpetas por defecto, este .jar puede ser ejecutado sin ningún problema y cumple la misma función que un .exe construido.

  src: contiene el código escrito este a su ves lo paquetes y las interfaces gráficas, escritas en java swing

- UI: Dentro podremos ver los archivos de Ilustrador que tienen el juego así como el logo personajes entre otros.

- Manual: Dentro de esta directorio podremos ver el manual de usuario del los jugadores.

- Informe: Dentro de este directorio se encuentra este documento que muestra el informe de la producción del proyectó.

  Javadoc: Construcción de los archivos .html del JavaDoc el cual cuenta con una documentación del proyecto y además tiene la descripción de cada unas de las clases del proyecto

- Compilado: Dentro podemos observar el juego medieval.exe el cual es el ejecutable para iniciar el juego.

URl: `https://gitlab.com/TEC-IC/IC-2101/I-17/G1/tree/master`

Conclusiones

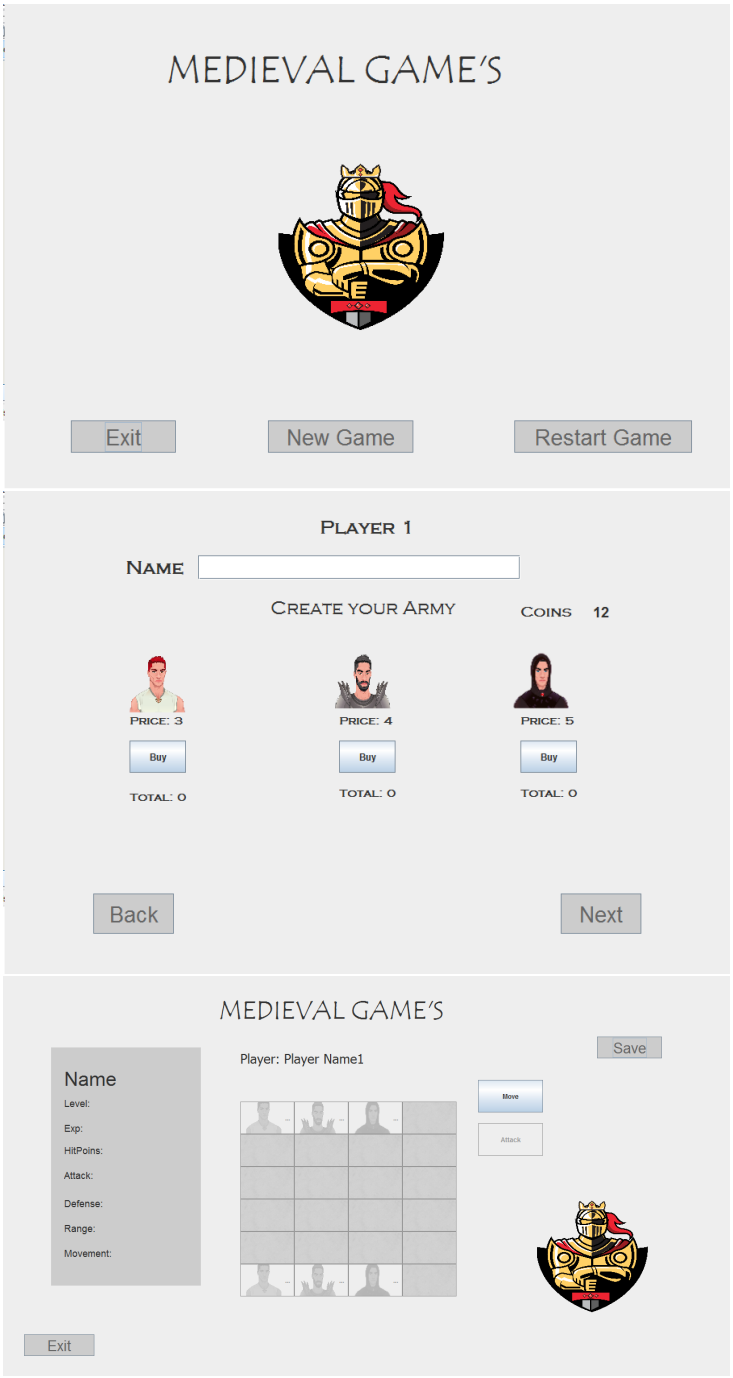
En este proyecto, hemos aprendido acerca de la programación orientada a objetos, ha descubierto una sintaxis Java que le permite crear objetos útiles y se ha familiarizado con un IDE que le ayuda a controlar su entorno de desarrollo. Sabe cómo crear y ejecutar objetos Java que pueden hacer una buena cantidad actividades, que incluyen hacer cosas diferentes en base a entradas diferentes. También sabe cómo hacer que sus aplicaciones admitan archivos JAR para que los otros desarrolladores las usen en sus programas y cuenta con algunas de las mejores prácticas básicas de programación Java en su haber.

Swing existe desde la JDK 1.1 (como un agregado). Antes de la existencia de Swing, las interfaces gráficas con el usuario se realizaban a través de AWT (Abstract Window Toolkit), de quien Swing hereda todo el manejo de eventos. Usualmente, para toda componente AWT existe una componente Swing que la reemplaza, por ejemplo, la clase Button de AWT es reemplazada por la claseJButton de Swing (el nombre de todas las componentes Swing comienza con "J").

Las componentes de Swing utilizan la infraestructura de AWT, incluyendo el modelo de eventos AWT, el cual rige cómo una componente reacciona a eventos tales como, eventos de teclado, mouse, etc... Es por esto, que la mayoría de los programas Swing necesitan importar dos paquetes AWT: java.awt.* y java.awt.event.*.

Ojo:Como regla, los programas no deben usar componentes pesados de AWT junto a componentes Swing, ya que los componentes de AWT son siempre pintados sobre los de Swing. (Por componentes pesadas de AWT se entiende Menu, ScrollPane y todas las componentes que heredan de las clases Canvas y Panel de AWT).

Anexos


MEDIEVAL GAME'S
Exit
New Game
Restart Game
PLAYER 1
NAME
CREATE YOUR ARMY
COINS 12
PRICE: 3
PRICE: 4
PRICE: 5
Buy
Buy
Buy
TOTAL: 0
TOTAL: 0
TOTAL: 0
Back
Next
MEDIEVAL GAME'S
Save
Name
Level:
Exp:
HitPoins:
Attack:
Defense:
Range:
Movement:
Player: Player Name1
Move
Attack
Exit