Manual técnico

Proyecto final Programación orientada a objetos

Integrantes:

Barrera Marroquín, Daniel Alexander 00023319

Villatoro Jurado, Javier Alexander 00199919

Aguirre Recinos, Estefany Elizabeth 00041319

Martínez Urbina, José Andrés 00019919

Índice

Aspectos generales

Objetivos del documento

Por medio de este documento se aspira a detallar los elementos que forman parte del software. Exponiendo y profundizando en el diseño creado, las herramientas y elementos utilizados.

Descripción general del proyecto

El programa cumple sus objetivos y aspiraciones de tomar el concepto de jugabilidad directamente del juego clásico Arkanoid (1986) y permitiendo al usuario experimentar parte de los elementos más destacables del juego original. Tanto en elementos como la integración de un sistema de puntajes como si de un juego de arcade se tratara hasta contar con características dentro de los niveles y su jugabilidad como lo son bloques los cuales requieren más de un golpe para ser destruidos.

Software utilizado

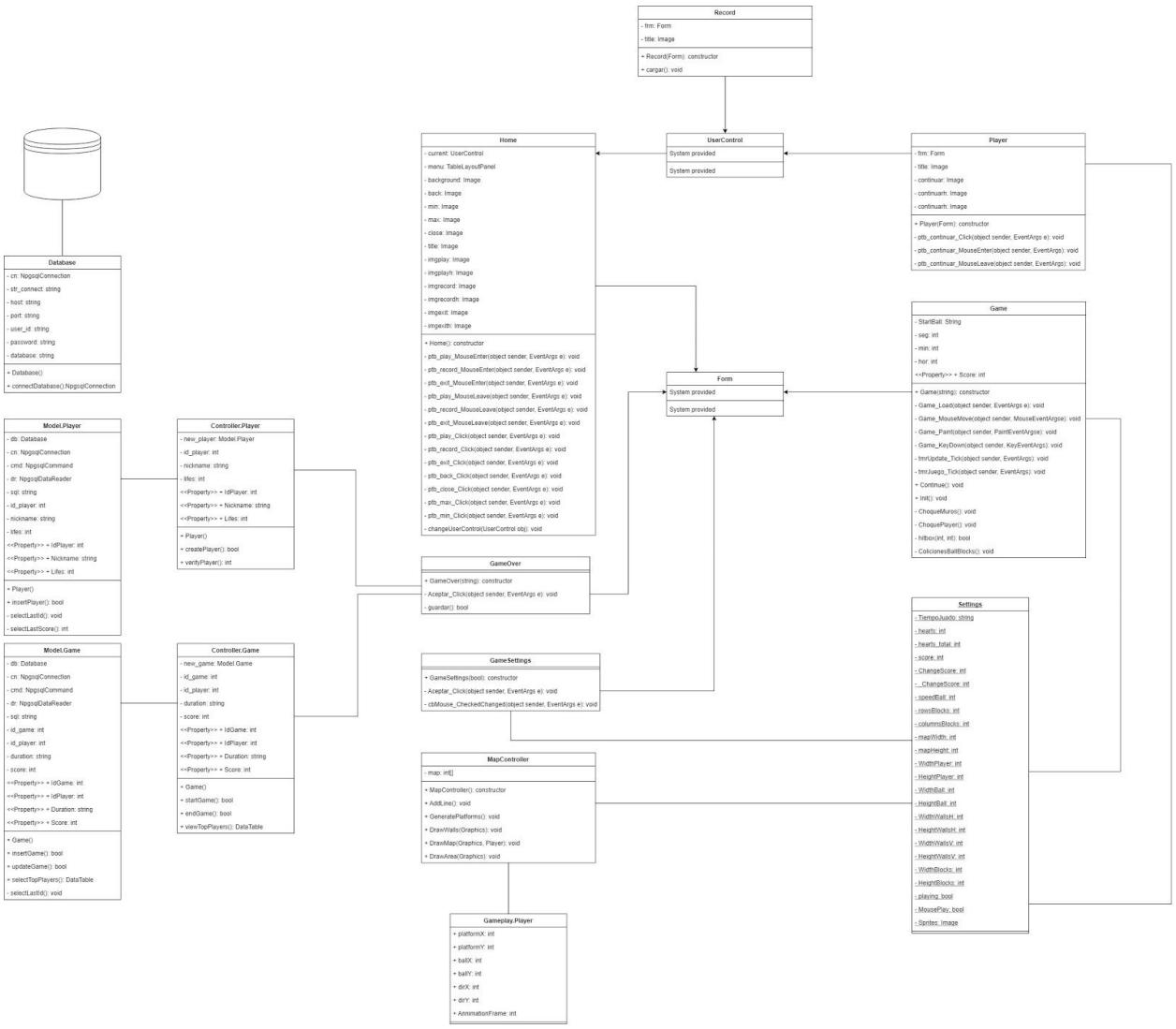
Para el desarrollo del programa se utilizó el IDE Rider de jetbrains utilizando c# como lenguaje de programación y junto con PostgresQL 11 se desarrolló la base de datos. Como elemento complementario, se utilizó Npgsql para la conexión de la base.

Modelos utilizados

UML diagrama de clases

A continuación se presenta el diagrama en el que se basa el diseño arquitectónico del código (siguiente página).

Dentro de la carpeta de documentación se encuentra una imagen donde observar con más detalle.



Dentro del diagrama de clases se pueden observar tanto las clases del modelo como las clases del controlador, la clase *Database* se conecta con la base de datos logrando que sean posibles las consultas que se realizan a esta. Dentro del modelo llamado *Game* se realizan las consultas para insertar, modificar y seleccionar información de las partidas realizadas y esta clase se conecta con

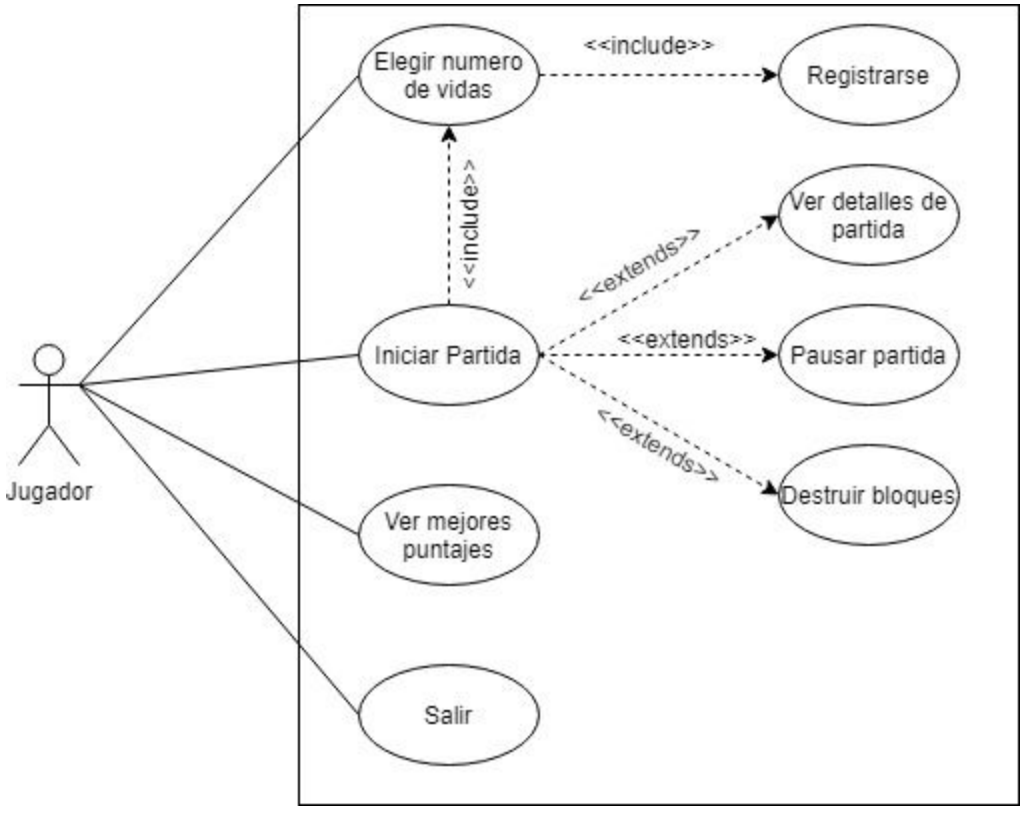
el controlador llamado *Game* en el que se encuentran los métodos que retornan las consultas principales del modelo. El modelo y controlador llamado *Player* cumplen la misma función que el anterior, solo que en este caso se trabaja con la información del jugador y no de la partida.

Home contiene las opciones principales del juego, por lo que en esta clase se compone principalmente por atributos y métodos para el diseño del formulario y para acceder a los UserControl que contiene estas opciones. La clase *Player* y *Record* son parte de estos UserControl que contienen los métodos para iniciar una partida y para visualizar el Top 10 de jugadores.

La clase *Game* contiene todos los métodos necesarios para que la jugabilidad sea posible y el diseño del juego se visualice de forma correcta, este form se conecta con las clases que componen la vista del juego. *Settings* contiene los atributos que configuran el diseño del juego, esta clase se conecta con la mayoría de los form y un UserControl, para que el movimiento del juego sea posible también se encuentra la clase *MapController* y Player que contienen los métodos para dibujar el diseño y los atributos para a animación de la bola.

Por último dentro del form *GameOver* se realiza la conexión con los controladores para poder almacenar la información de la partida una vez esta haya terminado.

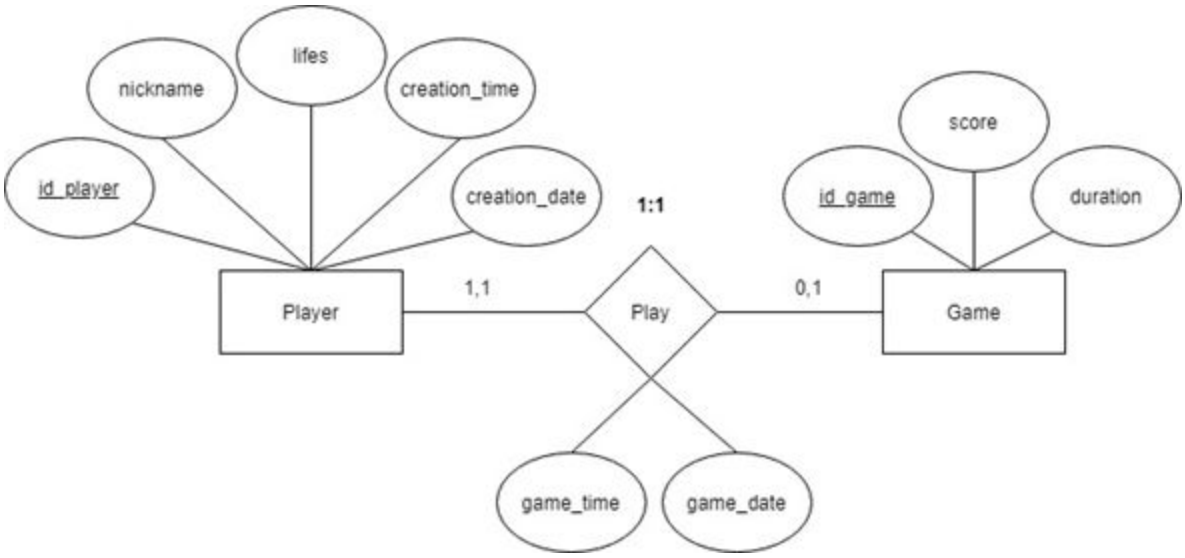
UML diagrama casos de usos



Dentro del diagrama se pueden observar las acciones principales que se cumplen en el videojuego. En cuanto un usuario quiere iniciar una partida este tiene la posibilidad de elegir el número de

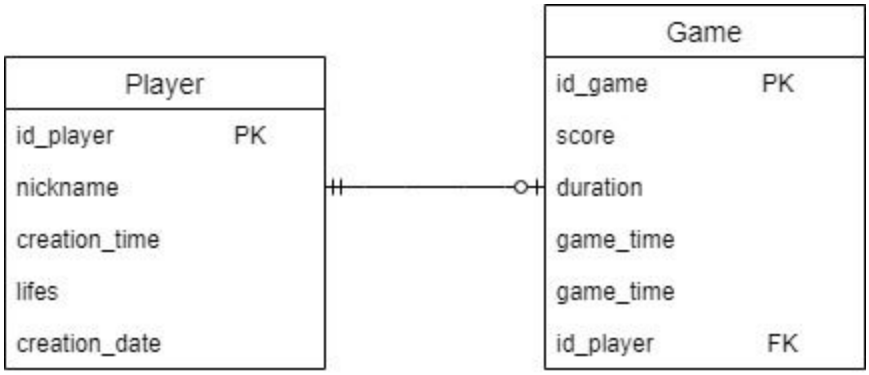
vidas con las que desea comenzar el juego, sin embargo debe antes registrarse en el juego, esto quiere decir que debe ingresar su nombre de usuario o registrar uno nuevo. Una partida consiste en destruir bloque y sumar puntaje en base a esto, una vez termina la partida el sistema le da la posibilidad al usuario de ver el detalle de la partida finalizada, tanto su puntaje como el tiempo que le llevó la partida. Otra de las acciones principales que puede realizar el jugador es visualizar los mejores puntajes registrados en el juego, solo se visualizan los mejores 10. El jugador tiene como opción salir de juego cuando ya no quiera seguir jugando.

Diagrama entidad relación extendido



Nuestra base de datos se compuso por dos entidades, *Player* y *Game*. Dentro de *Player* se almacena la información del jugador y dentro de *Game* la información de cada una de las partidas del jugador. Un jugador tendrá la posibilidad de jugar una partida a la vez.

Diagrama relacional



Este es el diagrama normalizado que se basó en el diagrama anterior en el que se puede observar mejor la relación de ambas entidades.

Conceptos técnicos

Implementación interfaz gráfica

El apartado gráfico del programa consta de 4 formularios los cuales sus nombres indican su funcionalidad y 2 controles de usuario que muestran las ventanas para agregar un nuevo jugador (Player.cs) y para visualizar los puntajes más altos (Record.cs).

Forms:

Game.cs

GameOver.cs

GameSettings.cs

Home.cs

UserControl:

Player.cs

Record.cs

Manejo de clases en modelo

Las siguientes clases forman parte del modelo del proyecto:

Database.cs

Game.cs

Player.cs

Plataforma base

• **Sistema Operativo**	Multiplataforma
• **Tecnologías**	Rider Versión: 2020.1.3
• **Lenguajes**	c#
• **Gestor de base de datos**	PostgresSQL

Nomenclatura

Abreviaturas

Las abreviaciones fueron realizadas y respetan el siguiente modelo <Abreviatura de tipo>_nombre, se utilizaron las siguientes:

• **PictureBox**	pbt
• **TableLayoutPanel**	tbly
• **Label**	lb
• **Timer**	trm
• **CheckBox**	cb
• **TextBox**	txt
• **NumericUpDown**	n
• **DataGridView**	dgv

Eventos y excepciones

Eventos

En el programa no se vio la necesidad de crear eventos personalizados, debido a que los UserControls manejan eventos independientes, comunicándose con el controlador correspondiente, gracias a la arquitectura Modelo Vista Controlador (MVC), tanto para la integración con Base De Datos y la Jugabilidad. Como dato extra, es importante mencionar que en el programa no se vio la necesidad de enviar al formulario principal eventos realizados en los userControls por la razón antes mencionada, la única funcionalidad implementada en los userControls fue cerrar el formulario abierto antes de empezar a jugar, esto se realizó enviando la variable *'this'* como parámetro al momento de instanciar el userControl *'Plalayer.cs'* que lo recibe en el constructor e iguala a una variable privada de tipo *Form*, y poder cerrar el formulario abierto desde el un userControl.

Excepciones

Gracias a sus nombres las excepciones son autoexplicativas. A continuación se enlistan las excepciones que forman parte del programa:

- **EmptyNicknameException.cs**
- **ExceededMaxCharactersException.cs**
- **ExceededMaxLifesException.cs**
- **NoLifesException,cs**
- **WrongKeyException.cs**
- **LessThanOneValueException.cs**