



## III Proyecto Programación Orientada a Objetos

II Semestre 2019

---

### Profesora

Samanta Ramijan

### Integrantes

Emmanuel Murillo Sanchez      2018201030

Jeison Sandi Mena      2016093589

Kevin Ceciliano      2019023828

## Tabla de Contenidos

<b>Tabla de Contenidos</b>	<b>1</b>
<b>Especificación del Proyecto</b>	<b>2</b>
Objetivos Formativos	2
Especificación del programa.	2
<b>Descripción del problema</b>	<b>5</b>
<b>Diagrama de Clases UML</b>	<b>6</b>
<b>Diseño del Programa:</b>	<b>7</b>
Decisiones de diseño, algoritmos usados, bibliotecas utilizadas, etc.	8
<b>Conclusiones Personales</b>	<b>16</b>
Emmanuel	16
Jeison	16
Kevin	17
<b>Autoevaluación</b>	<b>17</b>

## Especificación del Proyecto

### Objetivos Formativos

El principal objetivo de este proyecto es el de reforzar conocimientos en modelado y programación orientada objetos en Java. También se espera que el desarrollo de este proyecto permita a las y los estudiantes experimentar y adentrarse en el manejo de hilos de ejecución e interfaz gráfica de usuario con JavaFX. Se pretende así, que los estudiantes diseñen e implementen un juego de cuidado de mascotas digitales, estilo Tamagotchi, en la realización del mismo los estudiantes harán uso de los siguientes temas estudiados en clase:

- a) Modelado de Clases en UML.
- b) Entorno de Desarrollo (IDE) para programar en Java.
- c) Programación en Java.
  - 1) Objetos.
  - 2) Clases.
  - 3) Atributos.
  - 4) Métodos.
  - 5) Modificadores de visibilidad.
  - 6) Instanciación.
  - 7) Hilos de ejecución.
- d) Desarrollo de interfaces gráficas con JavaFX.

### Especificación del programa.

Existen una serie de indicadores que permiten saber el estado de la mascota, cada uno de estos indicadores puede tener un valor entero de entre cero y cinco. El estado de las mascotas definirá el puntaje de quien juegue.

Cada mascota tendrá sus horarios de comidas, así como sus necesidades particulares de cariño, higiene y salud. Por ejemplo Julius es un gato adulto que come cada día por la mañana a las 6am y de nuevo por la tarde a las 6pm. Es muy cariñoso, por lo que espera que lo acaricien cada dos horas, si esto se cumple el estará muy contento. Es un gato muy aseado, por lo que basta limpiar su espacio a la hora de la cena, una vez al día. Como Julius es un gato mayor, toma medicamentos tres veces al día, a las 6am, 4pm y 10 pm.

Al iniciar el programa, los indicadores son los siguientes:

Estado	Indicador
Felicidad	5
Salud	5
Suciedad	0
Hambre	0

Note cada animal podrá tener entre uno y tres tiempos de comidas y limpieza, y n cantidad de horarios para tomar medicamentos o ser acariciado. ( $0 \leq n < 24$ ).

Al atender a la mascota se modifican los indicadores, por ejemplo, al curar o limpiar a la mascota, su salud incrementa, al limpiarla además su suciedad baja a cero y su felicidad incrementa. Alimentar a las mascota supone que su nivel de hambre baja a cero. Acariciar a la mascota permite también hacerla más feliz.

Si no se cumplen los horarios y las necesidades de las mascotas, su estado se ve afectado, por ejemplo por cada hora que se pase desde la hora de comida del animal incrementa su nivel el hambre y disminuye su salud, por cada hora que se pase la hora de limpieza aumenta la suciedad y decrementa la salud. También si la mascota requiere cariño a una hora y no es acariciada, entonces baja su felicidad. Finalmente si la mascota necesita de un medicamento, su salud se disminuye por cada hora de

atraso para recibir el mismo.

El tiempo del programa estará acelerado, por lo que cada hora en el juego equivale a 5 segundos. Cada hora el programa verifica los horarios y necesidades de la mascota para actualizar los indicadores, a partir de estos da o quita puntos a quien juega según la siguiente tabla:

Estado	Puntos
Felicidad > 3	+10
Felicidad < = 3	-10
Salud = 5	+10
0 < Salud >= 2	-10
Suciedad >= 4	-10
Hambre < 4	+10

En caso de tener varias mascotas, existe un puntaje global que se ve afectado por los cuidados a todas las mascotas.

Note que los indicadores tienen un valor mínimo de 0 y uno máximo de 5, por lo que aún si la validación requiere restar el indicador y este ya está en 0, no se modifica.

De igual forma para indicadores que tengan un valor igual a 5 y se indique como necesario sumarlo, no se modifica.

Una vez que la salud esté en 0, quien juegue tendrá únicamente 2 horas para encargarse, o la mascota morirá y el juego se acabará. También termina el juego si la salud llega a 0 y la suciedad también.

## Descripción del problema

A través del presente proyecto, se establece como objetivo principal el diseño e implementación de un juego de cuidado de mascotas digitales estilo Tamagotchi, por lo tanto tomando en cuenta además que se espera que el desarrollo del mismo permita experimentar y adentrarse en el manejo de hilos de ejecución e interfaz gráfica de usuario con JavaFX, se plantea como problema la realización de dicha aplicación haciendo uso de las estrategias y herramientas anteriormente mencionadas, adicionalmente se espera de la misma forma a lo largo del desarrollo del proyecto hacer uso de otros de los contenidos expuestos previamente en el curso de Programación Orientada Objetos, como lo son el modelado de clases en UML, utilización de un entorno de desarrollo (IDE) para programar en Java y realización de algoritmos en el lenguaje de programación Java haciendo uso adecuadamente de objetos, clases, atributos, métodos, modificadores de visibilidad, e instanciación.

```

classDiagram
    class InGameController {
        -actualPet: Pet
        -actualPetName: String
        -previousPetName: String
        -petImage: ImageView
        -allTime: Label
        -allTime: Label
        -choiceBox: ChoiceBox
        -hungryProgressBar: ProgressBar
        -thirstProgressBar: ProgressBar
        -happinessProgressBar: ProgressBar
    }
    class Pet {
        -name: String
        -status: Status
        +Pet(name: String, status: Status)
        +getStatus(): Status
        +getPetScore(): String
    }
    class PetSelectionController {
        -petDog: ImageView
        -stage: Stage
        -firstTime: Boolean
        -refreshPetImage: Button
        -refreshPetScore: Button
    }
    class GameCentral {
        +player: Player
        +stage: Stage
        +firstTime: Boolean
        +refreshGameData(): void
        +refreshPetStatus(): void
        +refreshPetScore(): void
        +refreshPetImage(): void
        +refreshPetScore(): void
        +refreshPetImage(): void
        +refreshPetScore(): void
    }
    class Player {
        -name: String
        -playerScore: int
        -pets: ArrayList<Pet>
    }
    class GameCritic {
        -fileName: String
        +getFileName(): String
        +getPetScore(): String
        +getPetScore(): String
        +getPetScore(): String
    }
    class PetSelectionController {
        -petDog: ImageView
        -stage: Stage
        -firstTime: Boolean
        -refreshPetImage: Button
        -refreshPetScore: Button
    }

    InGameController --> Pet
    InGameController --> PetSelectionController
    PetSelectionController --> GameCentral
    GameCentral --> Player
    Player --> GameCritic
    GameCritic --> PetSelectionController
    PetSelectionController --> PetSelectionController
  
```

**UML Class Diagram Details:**

- InGameController**
  - Attributes: `-actualPet: Pet`, `-actualPetName: String`, `-previousPetName: String`, `-petImage: ImageView`, `-allTime: Label`, `-choiceBox: ChoiceBox`, `-hungryProgressBar: ProgressBar`, `-thirstProgressBar: ProgressBar`, `-happinessProgressBar: ProgressBar`
  - Methods: `+initGame()`, `+impulse()`, `+accelerate()`, `+decelerate()`, `+turnLeft()`, `+turnRight()`, `+refreshGame()`, `+refreshScore()`, `+showPetSelectionScreen()`
- Pet**
  - Attributes: `-name: String`, `-status: Status`
  - Methods: `+Pet(name: String, status: Status)`, `+getStatus(): Status`, `+getPetScore(): String`
- PetSelectionController**
  - Attributes: `-petDog: ImageView`, `-stage: Stage`, `-firstTime: Boolean`
  - Methods: `+refreshPetImage()`, `+refreshPetScore()`, `+refreshPetImage()`, `+refreshPetScore()`
- GameCentral**
  - Attributes: `+player: Player`, `+stage: Stage`, `+firstTime: Boolean`
  - Methods: `+refreshGameData()`, `+refreshPetStatus()`, `+refreshPetScore()`, `+refreshPetImage()`, `+refreshPetScore()`, `+refreshPetImage()`, `+refreshPetScore()`
- Player**
  - Attributes: `-name: String`, `-playerScore: int`, `-pets: ArrayList<Pet>`
- GameCritic**
  - Attributes: `-fileName: String`
  - Methods: `+getFileName(): String`, `+getPetScore(): String`, `+getPetScore(): String`, `+getPetScore(): String`
- PetSelectionController (Bottom)**
  - Attributes: `-petDog: ImageView`, `-stage: Stage`, `-firstTime: Boolean`
  - Methods: `+refreshPetImage()`, `+refreshPetScore()`, `+refreshPetImage()`, `+refreshPetScore()`

**Relationships:**

- Associations:**
  - `InGameController` to `Pet` (1 to 1)
  - `InGameController` to `PetSelectionController` (1 to 1)
  - `PetSelectionController` to `GameCentral` (1 to 1)
  - `GameCentral` to `Player` (1 to 1)
  - `Player` to `GameCritic` (1 to 1)
  - `GameCritic` to `PetSelectionController` (1 to 1)
  - `PetSelectionController` to `PetSelectionController` (1 to 1)
- Aggregations:**
  - `GameCentral` aggregates `Player` (indicated by a hollow diamond on the `GameCentral` side).
  - `Player` aggregates `GameCritic` (indicated by a hollow diamond on the `Player` side).
- Generalization:**
  - `PetSelectionController` (Bottom) is a generalization of `PetSelectionController` (Top) (indicated by a hollow triangle arrow pointing from the bottom to the top).

## Diseño del Programa:

Las decisiones entorno a la implementación y diseño del programa requerido a través del presente proyecto, fueron llevadas a cabo con base en los requerimientos establecidos previamente en las instrucciones proporcionadas para la elaboración del mismo, lo anterior con el fin de cumplir con lo solicitado a través de la rúbrica y el cumplimiento de los objetivos planteados mediante la elaboración del mismo.

Partiendo de lo anterior, algunos de los aspectos que se considera importante resaltar, son la utilización de el entorno de desarrollo (IDE) conocido IntelliJ IDEA, instalado en el sistema operativo windows 10, la decisión de utilizar este entorno de desarrollo fue impulsada por el buen desenvolvimiento de dicha aplicación con el lenguaje de programación Java y el deseo del equipo por explorar el uso de dicho entorno de desarrollo a través de la ejecución del presente proyecto. En el caso del sistema operativo, si bien todos los miembros del equipo coinciden es que hubiese sido preferible haber utilizado Linux como sistema operativo, la decisión de elegir windows se dió más por un tema de disponibilidad de recursos de los integrantes del grupo previo a la ejecución de dicho trabajo.

Por otra parte y no menos importante, como ya se indicó anteriormente, dado que la elaboración de dicho proyecto lo requiere se llevará a cabo la implementación de hilos de ejecución y de interfaz gráfica de usuario con JavaFX.

A continuación, se detallan algunos aspectos más concretos, en relación con el código o programa desarrollado, relacionados con la utilización de objetos, clases, atributos, métodos, entre otros.



## Decisiones de diseño, algoritmos usados, bibliotecas utilizadas, etc.

De acuerdo con lo anterior, en cuanto a las decisiones propias de la elaboración del código, a lo largo del diseño e implementación del algoritmo se determinó que para la realización de este se utilizarían las clases con los respectivos algoritmos mostradas a continuación:

### InGameController

```
+initialize ( ): void  
+limpiar(): void  
+acariciar():void  
+curar():void  
+alimentar():void  
+getActualPet():void  
+refresh(): void  
+refreshScene(): void  
+showPetSelectionScreen (event:ActualEvent): void
```

### Time

```
+getIntHours(): int  
+getTimeArrayJSON(): String  
+getTime(): String  
+setIntDay(intDay: int): void  
+setIntHours(intHours: int):  
void  
+setIntMinutes(intMinutes: int):  
void  
+startReloj():void  
+addMinute():void  
+updateDay():void
```

```
+updateHours():void  
+updateMinutes():void
```

### Pet

```
+Pet()  
+Pet(name: String)  
+Pet(name:String,  
status: Status)  
+getName():String  
+getstatus():Status  
+getPetJSON():String
```

### GameFile:

```
+fileName():String  
+setFileName(fileName:String):void  
-readStrFromFile(str: String): String  
+writeStrToFile(str: String): void  
+saveGameData(player: Player, xd: String)  
+loadGameData(): Player  
+saveGameData(): void
```

### PetSelectionController

```
+initialize(): void
+setPetsToSelect(): void
+defaultDog(): Pet
+defaultCat(): Pet
+defaultRabbit():Pet
+defaultBird(): Pet
+showInGameScreen(): void
```

## Pet

```
+Player(name: String)
+Player(name: Strin, playerScore:int,
pets:ArrayList<Pet>, tm:time)
+Player(name: Strin,
pets:ArrayList<Pet>, tm:time)
+addPet(pet: Pet): void
+getName(): String
+setName(name:String):void
+getPlayerScore(): int
+setPlayerScore(playerScore: int): void
+getPets():ArrayList<Pet>
+setPets(pets: ArrayList<Pet>): void
+getTm(): Time
+setTime(tm: Time): void
+getPetByName(petName:String):Pet
+getPlayerJSON(): String
```

## GameCentral

```
+getPlayer: Player
+setPlayer(player: Player): void
+setStage(stage: Stage): void
```


```
+isFirstTime(): Boolean
+setFirstTime(firstTime: Boolean): void
+refreshGameData(): void
+refreshPetStatus(status: Status): void
+refreshPetScore(status: Status)
+curar(pet: Pet): void
+limpiar(pet Pet): void
+acariciar(pet:Pet): void
+alimentar(pet:Pet): void
+delayChecker(status: Status)
+startRefreshThread(): void
+showScene(fxmlName: String,
title:String):void
```

## PetSelectionController

```
+initialize():void
+setPetsToSelect():void
+defaultDog(): Pet
+defaultCat(): Pet
+defaultRabbit(): Pet
+defaultBird(): Pet
+showInGameScreen(): void
```

## Status

```
+Status(int happiness, int health, int dirt, int hungry, int
petScore, int curarDelay, int limpiarDelay, int
acariciarDelay, int alimentarDelay, ArrayList<Integer>
affectionHours, ArrayList<Integer> eatingHours,
ArrayList<Integer> hygieneHours, ArrayList<Integer>
medicationHours)
+getHappinness(): int
+getHealth():void
+getDirt(): int
```



```
+getHungry(): int  
+getPetScore(): int  
+incPetScore(n:int):void  
+decPetScore(n:int):void  
+incHappiness(n:int):void  
+decHappiness(n:int):void  
+incHealth(n:int):void  
+decHealth(n:int):void  
+incDirt(n:int):void  
+decDirt(n:int):void  
+incHungry(n:int):void  
+decHungry(n:int):void  
+getCurarDelay(): int
```

```
+setCurarDelay(curarDelay:int):void  
+getLimpiarDelay(): int  
+setLimpiarDelay(limpiarDelay:int):void  
+getAcariciarDelay():int  
+setAcariciarDelay(acariciarDelay):void  
+getAlimentarDelay():int  
+setAlimentarDelay(alimentarDelay:int): int  
+getAffectionHours():ArrayList<Integer>  
+getEatingHours()ArrayList<Integer>  
+getHygieneHours():ArrayList<Integer>  
+getMedicationHours():ArrayList<Integer>  
+getStatusJSON():String
```

## Análisis de resultados:

En relación con los requerimientos y objetivos de elaboración del presente proyecto, los cuales fueron expuestos al inicio de este documento, a continuación se realizará un análisis general de los resultados obtenidos a través de la experiencia en la realización del mismo. De manera preliminar es importante resaltar que al llevar a cabo la ejecución de cada uno de los objetivos previamente establecidos el resultado en general fue bastante positivo y aunque el equipo es consciente de que probablemente en la mayoría de ocasiones haya oportunidades de mejora, dado que los resultados fueron positivos no se incluye ningún objetivo como no logrado o no alcanzado.

Como primer objetivo formativo, se estableció reforzar conocimientos en modelado y programación orientada objetos en Java. En el caso de este objetivo, el mismo fue cumplido exitosamente y realmente no hubo ningún problema al ejecutarlo, de acuerdo con la experiencia al realizar dicho paso, queda completamente comprobado que el diseño de clases a través de diagramas y la utilización en general de los métodos de modelado facilitó la escritura del código.

Por su parte el realcion con el objetivo de experimentar en el manejo de hilos de ejecución e interfaz gráfica de usuario con JavaFX, tampoco hubo ningún problema y como se puede verificar en el código, la utilización de los hilos fue realmente muy exitosa ya que como lo requería la aplicación desarrollada permite estar ejecutando más con JavaFx, sin duda el uso de esta plataforma gráfica fue todo un éxito ya que gracias a la gran cantidad de opciones gráficas y de edición con la que cuenta, nos permite realizar la aplicación de una forma visualmente agradable y a la vez relativamente sencilla.

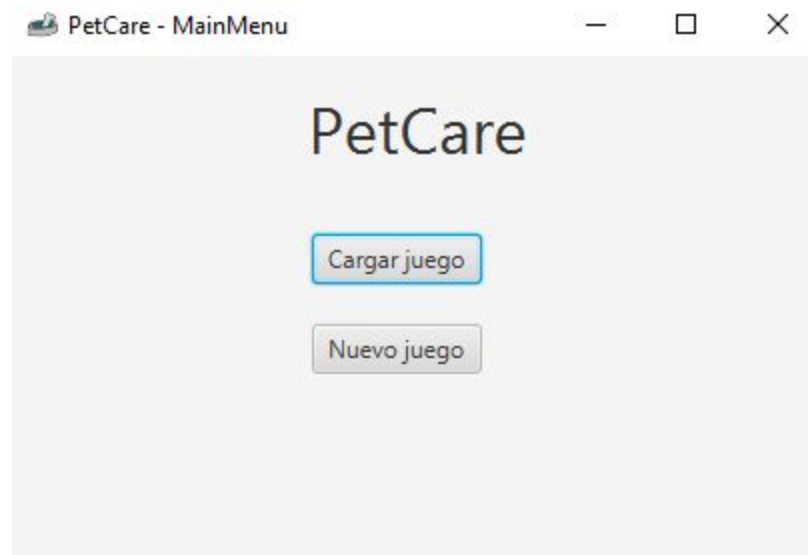
Finalmente, en cuanto al objetivo de implementar un juego de cuidado de mascotas digitales, estilo Tamagotchi este fue alcanzado con éxito y la aplicación Pet Care desarrollada por el equipo logró cumplir con dicha finalidad. En este sentido, la utilización de objetos, clases, atributos, métodos, entre otros, fue alcanzada satisfactoriamente.

## Manual de usuario: Instrucciones de uso:

Para la compilación y ejecución del programa se implementó el formato .jar el cual para ejecutar se abre como un ejecutable Java. Esto permite mantener el proyecto empaquetado y que pueda compilarse y ejecutarse fácilmente.

En general, el funcionamiento de la aplicación desarrollada denominada Pet Care podría catalogarse como realmente muy sencilla, a continuación se detallan algunos de los pasos requeridos para su ejecución y uso adecuado.

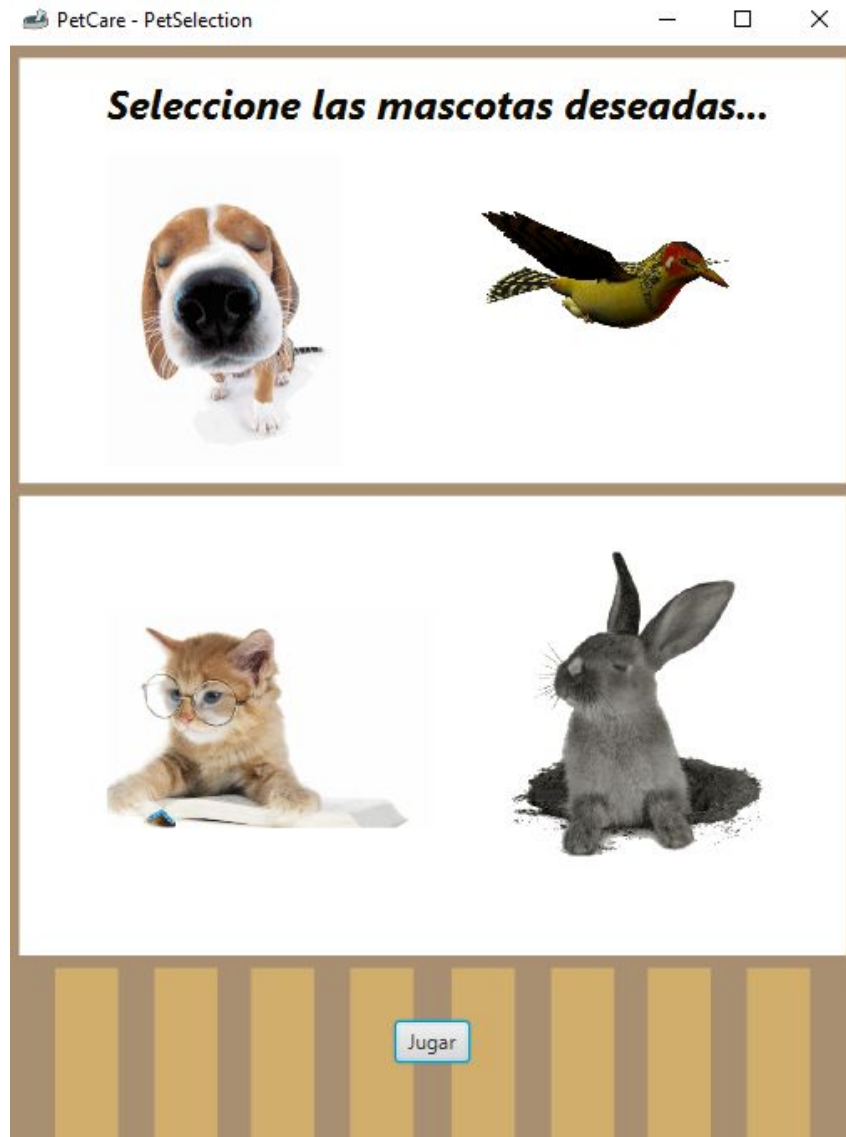
Primeramente la aplicación ofrece la posibilidad de cargar un juego previamente guardado o de iniciar una partida nueva completamente desde cero, en este caso lo únicamente debe hacerse clic en la opción deseada que se nos muestra en pantalla como se observa en la siguiente imagen:



Seguidamente, para cualquiera de las opciones elegidas, ya sea cargar un juego previamente iniciado, o crear un juego nuevo, se debe digitar el nombre del jugador, en la ventana que se muestra a continuación:

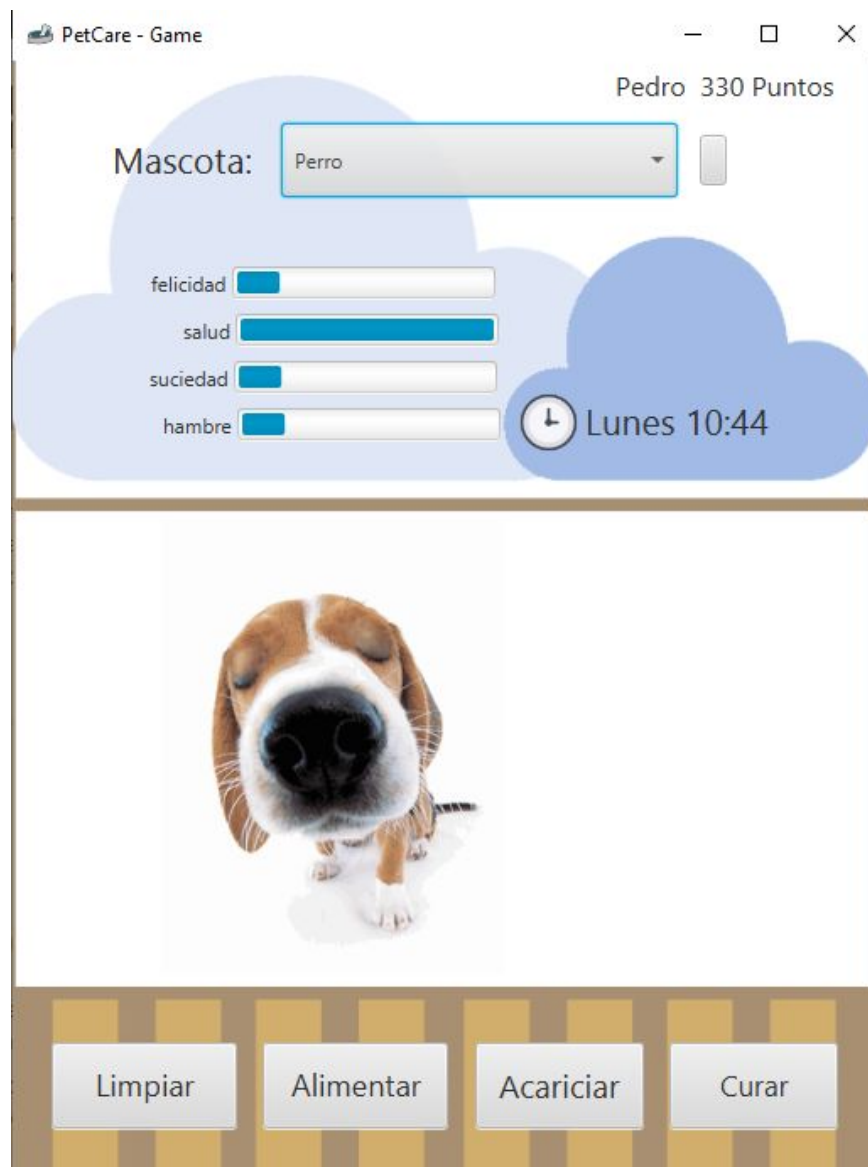


En caso de que se haya cargado una partida, las mascotas con las que cuenta el jugador serán cargadas de forma automática. Por su parte, en el caso de las partidas nuevas luego de ingresar el nombre del jugador y hacer clic en partida nueva, se desplegará la pantalla siguiente, la cual nos permitirá elegir las mascotas deseadas haciendo clic sobre cada una de ellas y finalmente haciendo clic en el botón jugar, como se muestra a continuación:



Una vez ya iniciado el juego, como se observa en la siguiente imagen, en la parte superior de la interfaz gráfica se muestran algunos datos importantes de la partida, como el nombre del jugador, su puntaje, el tiempo del juego, los diferentes indicadores de estado de la mascota y la opción para intercambiar entre cada una de ellas de acuerdo con la cantidad elegida por el jugador.

Adicionalmente, en la parte inferior se encuentran los botones que permiten modificar los indicadores de estado correspondientes a cada una de las mascotas, con el fin de mantener la mascota en óptimas condiciones, lo cual es el objetivo principal de juego..





## Conclusiones Personales

### Emmanuel

A nivel personal, la experiencia adquirida a través de la elaboración de este proyecto ha sido muy enriquecedora, sobre todo tomando en cuenta que el llevar a la práctica algunos contenidos como el uso de hilos de ejecución y la utilización de interfaz gráfica con JavaFX ha sido a mi parecer una aventura realmente muy interesante y divertida. Por su parte el tipo de aplicación o programa realizado también ha representado un reto importante, el cual fue gratificante alcanzar y lo más valioso de esto, ha sido la experiencia adquirida a través de todas aquellas pruebas tanto certeras como muchas veces erróneas que fue necesario superar para alcanzar el producto final.

Adicionalmente, al igual que en caso de los proyectos anteriores la retroalimentación del trabajo en equipo en el presente proyecto es otro aspecto el cual a nivel personal me atrevo a destacar, ya que sin duda la comunicación efectiva y la disposición del equipo para cumplir con los requerimientos y deberes de trabajo fueron un punto alto que permitió la conclusión exitosa del mismo.

### Jeison

Durante el desarrollo del proyecto al implementar conceptos analizados en clase como la implementación de interfaz gráfica y el uso de hilos para gestionar los procesos, se logró una mejor comprensión sobre estos recursos y a nivel personal una perspectiva que permite mayor flexibilidad para implementarlos en otras circunstancias en las cuales con el uso de estas herramientas pueden desarrollarse de forma más atractiva y eficiente.

Tal como ha sucedido anteriormente la retroalimentación dentro del grupo de trabajo fue muy acertada e importante durante el proceso, lo cual a mi percepción permitió implementar la versión más eficiente dentro de las posibilidades del grupo.

## Kevin

Realizar este proyecto fue, para mi persona, de muchas formas enriquecedor, desde la experiencia que se gana al trabajar en grupo, escuchar los comentarios de cada miembro del grupo para mejorar el trabajo, toma de decisiones de forma grupal, hasta el poder poner en práctica todos los conceptos visto en el curso de Programación Orientada a Objetos.

Fue interesante implementar por primera vez una Interfaz Gráfica de Usuario, la inicio fue difícil entender cómo funcionaba el JavaFX pero poco a poco fue entendiendo mejor y fue una experiencia gratificante el elaborar el proyecto

## Autoevaluación

### Criterio #1

Cumplimiento con todas las tareas en el plazo definido por el grupo, fortaleciendo la comunicación y uso de habilidades blandas para lograr realizar un buen trabajo en equipo.

Evaluación individual			Calificación de compañeros		
Integrante	Porcentaje de participación	Comentario	Emmanuel	Kevin	Jeison
Emmanuel	30%	La organización de la ejecución del trabajo fue eficiente.	N/A	10	10
Kevin	40%	El trabajo en equipo a la hora de realizar este	10	N/A	10

		proyecto fue impecable			
<b>Jeison</b>	30%	La organización de las tareas fue buena.	10	10	N/A
<b>Total</b>	100%				

## Criterio #2

Participación activa en la elaboración del trabajo en general, aporte de observación o comentarios enriquecedores tanto para el trabajo como para la experiencia de los otros miembros del equipo.

Evaluación individual			Calificación de compañeros		
Integrante	Porcentaje de participación	Comentario	Emmanuel	Kevin	Jeison
Emmanuel	33%	La retroalimentación entre los miembros del equipo fue un punto alto.	N/A	10	10
Kevin	34%	Los compañeros fueron activos brindando comentarios y formas de mejorar el trabajo	10	N/A	10
Jeison	33%	Los comentarios ayudaron a hacer mejoras al programa.	10	10	N/A
<b>Total</b>	100%				

### Criterio #3

Desarrollo adecuado de conceptos vistos a lo largo de curso de Programación Orientada a Objetos, dominio de los principios y procesos asociados a dichos conceptos e implementación correcta en el desarrollo del proyecto.

Evaluación individual			Calificación de compañeros		
Integrante	Porcentaje de participación	Comentario	Emmanuel	Kevin	Jeison
Emmanuel	40%	Hubo aportes relacionados con este criterio de todos los miembros del equipo.	N/A	9	9
Kevin	30%	Se aplicaron conceptos vistos en el curso de forma adecuada con la implementación de la solución al problema dado	10	N/A	10
Jeison	30%	Los temas se desarrollaron e implementaron adecuadamente.	10	10	N/A
Total	100%				

### Criterio #4

Disposición para realizar correcciones o aceptar sugerencias con el fin de alcanzar una mejor aplicación de los conceptos necesarios para la elaboración del proyecto.

Evaluación individual			Calificación de compañeros		
Integrante	Porcentaje de participación	Comentario	Emmanuel	Kevin	Jeison
Emmanuel	34%	La disposición por entregar lo mejor de cada uno, facilitó el desarrollo de este criterio.	N/A	9	9
Kevin	33%	Los compañeros reciben los consejos dados por el equipo de trabajo para mejorar el proyecto	10	N/A	10
Jeison	33%	La retroalimentación fue positiva y todos contribuyeron en este aspecto.	10	10	N/A
Total	100%				

## Criterio #5

Aporte de tiempo y retroalimentación a través de la búsqueda de información en fuentes externas con la finalidad de ampliar el conocimiento en el tema y por ende poder aplicar mejoras en el proyecto.

Evaluación individual			Calificación de compañeros		
Integrante	Porcentaje de participación	Comentario	Emmanuel	Kevin	Jeison
Emmanuel	30%	Los aportes dichas fuentes en la resolución de problemas fue vitales.	N/A	10	10
Kevin	30%	Cada miembro del grupo investiga y se adentra en el tema que le toca llevar a cabo	10	N/A	10
Jeison	40%	La investigación grupal fue eficiente y oportuna.	10	10	N/A
Total	100%				

\*\* La escala de calificación indica 1 como la nota inferior o más baja y 10 como la nota excelente.