Un controlador para dispositivos móviles adaptado al usuario utilizando algoritmos heurísticos y aprendizaje de máquina

Renato Roberto Chavez Urday

Orientador: Yván Jesús Túpac Valdivia

Tesis presentada a la Escuela Profesional de Ciencia de la Computación como parte de los requisitos para obtener el Título Profesional de Lic. en Ciencia de la Computación.

UCSP- Universidad Católica San Pablo
Mayo de 2022

Resumen

Al usar un aplicativo, el usuario espera una interacción fácil e intuitiva. Los controladores en actuales para dispositivos móviles suelen tener una configuración predeterminada de botones, juegos o aplicativos diferentes requieren diferentes botones o exigen diferentes métodos de interacción. Además, el estilo del jugador varía según características personales o experiencias de juego pasadas. En este documento se propone un *layout* o controlador virtual para dispositivos móviles capaz de adaptarse al comportamiento del usuario a partir de la recopilación y análisis de los *touches* de un usuario. Nuestra implementación adapta dinámicamente el tamaño, forma, color, transparencia, grosor de borde y sombras, a partir de heurísticas que son ejecutadas en un servidor, para evitar que el dispositivo del usuario realice el cálculo de la nueva renderización del *layout*. A través de la experimentación este documento demuestra la utilidad y efectividad de la adaptación implementada, mejorando la comodidad y desempeño del usuario. Además nuestra propuesta es fácilmente integrable debido a la compatibilidad del API implementado. Esperamos que nuestra propuesta reduzca el trabajo de diseño y mejora la experiencia del usuario en juegos o aplicativos móviles.

Índice general

Índice de figuras

Glosario

Avatar Identidad virtual que escoge el usuario de una computadora o de un videojuego para que lo represente en una aplicación o sitio web.

Controlador Dispositivo físico o virtual que sirve para controlar un programa o software.

Layout Distribución de los componentes de un controlador.

***Touch* / toque** Acción de tocar y reconocer las posiciones de los dedos en el dispositivo del usuario.

Capítulo 1

Introducción

1.1. Motivación y Contexto

Con el avance de la tecnología en videojuegos o aplicativos para dispositivos móviles, en estos días se cuenta con más formas de controladores virtuales o *layouts* que antes, como por ejemplo, los botones de movimiento de un videojuego o los teclados virtuales de los celulares. La calidad y fluidez de estos nuevos diseños y controles definirán la percepción del usuario acerca de su experiencia en el aplicativo. Existen ciertos diseños tradicionales para controladores virtuales en dispositivos móviles capaces de dar la funcionalidad adecuada al usuario, sin embargo, crear una distribución apropiada del controlador personalizada al jugador o al tipo de juego puede crear una impresión memorable del aplicativo. Esta conexión directa entre la experiencia del usuario y el diseño del controlador virtual es fácil de notar debido a que los controles del aplicativo o juego son el nexo entre lo que el usuario quiere hacer y lo que puede hacer en el ambiente virtual.

Los dispositivos móviles han abierto un nuevo paradigma de interfaz de usuario, dando mayor libertad al diseñar los controladores de los aplicativos, debido a que son virtuales. Además son muy dinámicos, ya que tienen diferentes componentes y dispositivos a su disposición tales como el bluetooth, giroscopio, o cámara. Al ser un medio virtual, no es necesario utilizar todas estas funcionalidades al mismo tiempo, solo las suficientes para que el controlador se acople al juego. Al existir diferentes tipos de aplicativos y funcionalidades, propiedades como la distribución de botones, textos y relación de los componentes con sus funcionalidades se complican. Debemos buscar que el controlador virtual sea cómodo, personalizado e intuitivo para el usuario.

1.2. Planteamiento del Problema

Al ofrecer funcionalidades innecesarias al usuario de un aplicativo móvil, que el diseño no sea personalizado al usuario así como no tener implícita la relación entre el diseño de un componente con la funcionalidad que representa, generamos diseños que no serán útiles. Para evitar esto debemos generar distribuciones adecuadas a un aplicativo

o usuario en específico, que se acople al comportamiento del usuario en tiempo real. De esta manera se adaptará de manera inherente a las características únicas que presente el usuario o el aplicativo.

Esta personalización o adaptación de los controladores debe ser considerada en el diseño de los *layouts* virtuales.. Por ejemplo, el tamaño de la mano o dedos de los jugadores puede influir en su comodidad para presionar los botones. Si tiene los dedos muy largos, presionara con más frecuencia botones más cercanos al centro. Si son más cortos, usará los pulgares en las esquinas. Otro ejemplo es considerar si una persona es zurda o diestra. Si es zurda presionara con más frecuencia el lado izquierdo de la pantalla y si es diestro el lado derecho de la pantalla, por lo que deberíamos enfocarnos en que resalte mas ese lado, o brindar los botones o componentes más importantes dependiendo del lado de la pantalla más usado.

En los ejemplos anteriores, se pudo analizar el comportamiento a través de los *touches* del usuario a la pantalla. Es por eso que en este trabajo utilizamos los *touches* como fuente de información para analizar el comportamiento del usuario. Los componentes de los controladores tienen varias características a ser considerados para modificarse, tales como el color, tamaño, posición, tonalidad y forma. Debemos crear un *layout* que considere estas características. Además que el comportamiento del usuario varía con el tiempo, por lo que el diseño del controlador debe ser dinámico.

1.3. Objetivos

En este trabajo buscamos implementar una API capaz de adaptar de manera remota y automática un controlador virtual de un dispositivo móvil. Así como proponer diferentes heurísticas para modificar la posición, tamaño, color, bordes y sombras de los componentes del controlador.

1.3.1. Objetivos Específicos

- Proponer diferentes heurísticas para mejorar la distribución y propiedades de los componentes de un *layout.*
- Implementar una API capaz de adaptar un *layout* al comportamiento del usuario.
- Realizar pruebas de nuestra propuesta para medir su usabilidad y practicidad.

1.4. Organización

El restante de este trabajo de plan de tesis está organizado de la siguiente manera:

- En el capítulo 2 se desarrolla el marco teórico.

- En el capítulo 3 se revisa el estado del arte acerca de los métodos de adaptación de *layouts* existentes.
- En el capítulo 4 se detalla la propuesta de este trabajo, detallando las heurísticas utilizadas.
- En el capítulo 5 se explica los experimentos y resultados obtenidos.
- El el capítulo 6 se dan las conclusiones y se plantean trabajos futuros.

Capítulo 2

Marco Teórico

Al usar un aplicativo móvil, una de las principales características que definirá la percepción que el usuario tendrá sobre su experiencia, es la calidad y fluidez de los controles del juego. Por tanto, en toda la zona de utilización, un factor importante para la experiencia de juego es el controlador o el esquema de control. Además los controles deben ser intuitivos, lo que permite a un usuario nuevo comenzar a interactuar inmediatamente sin dificultades y proporcionar una profunda interacción, con una amplia gama de posibles acciones de la forma más sencilla. Por último, deben ser adaptables y de baja utilización de recursos para el dispositivo del usuario.

2.1. *Layout* Adaptable

Un *layout* adaptable es un sistema de software interactivo que mejora su capacidad de interactuar como usuario en función de su experiencia parcial. [Stewart, 2013] desarrollaron los modelos que obtienen los (*screen touches*) y usan esta información para lidiar con el traspaso de control entre el usuario y el sistema. Demuestran un navegador de mapas de dedos, que desplaza el mapa a un punto de interés cuando la entrada del usuario es incierta. Manteniendo el mismo objetivo, pero de una manera diferente, [Weir, 2012] utilizaron un enfoque de aprendizaje automático para aprender modelos de entrada táctil específicos del usuario para aumentar la precisión táctil en los dispositivos móviles. Propusieron registrar datos del punto de contacto previsto, basado en el comportamiento táctil histórico de un usuario específico. Gracias a su trabajo, podemos darnos cuenta de la importancia de registrar los *touches* de un usuario para analizar su comportamiento.

De manera similar, [Bi, 2013] conceptualizan la entrada táctil con el dedo como un proceso incierto, y utilizaron un criterio estadístico de selección de objetivos. Mejoraron la precisión táctil utilizando un criterio de toque bayesiano y disminuyeron considerablemente la tasa de error. Sin embargo, incluso mejorando la precisión en un nivel superior, el usuario sigue perdiendo los botones y la interfaz necesita calcular el objetivo previsto. Por esto es importante relacionar los *touches* registrados a un componente o botón, para darle un sentido a la acción del usuario. Esto nos ayuda a adaptar la interfaz para que los usuarios puedan evitar errores. Para lograrlo, utilizamos los datos sobre correctos y

pulsaciones incorrectas de botones virtuales para cambiar diferentes propiedades del botón, y retroalimentar este dinamismo del rate de *touches* exitosos en el registro de un componente.

En [Zamith, 2013] se propuso un método para construir un controlador personalizable basado en un dispositivo móvil. En dicho trabajo se presentó un controlador móvil creado y diseñado por el desarrollador del juego adaptable al juego mas no al usuario. Además, no incluye ningún tipo de adaptación para mejorar la interfaz diseñada. Por lo que utilizamos su modelo para la relación entre el controlador y el aplicativo, así como para mejorar el *layout* desde un servidor remoto. En [Zamith, 2013] se hace un primer intento de *layout* adaptable, de ajustes simples, moviendo botones de acuerdo a los mapas de calor generados por los *touches*. La mayor limitación en esta investigación es que no se utiliza un algoritmo inteligente para los movimientos, por lo que un *touch* en el lado interior de un cuadrante tendrá el mismo peso que un toque en el exterior área del cuadrante, creando un resultado sesgado.

En el presente trabajo se propone un enfoque más sofisticado y novedoso, que registra todos los *touches* en la pantalla, asignándoles un *rate* de éxito y un componente objetivo. Desarrollamos un enfoque capaz de monitorear automáticamente el comportamiento del usuario, recopilando todos los datos de entrada durante la interacción del usuario y registrándose en una base de datos con la tasa de error de cada *touch* al intentar tocar un botón. Con esta información fue posible realizar varios análisis estadísticos así como comparaciones y determinar si el *layout* adaptable realmente aumenta el desempeño, comodidad o efectividad del controlador virtual.

2.2. API de renderización, asincronicidad y escalamiento

Respecto a cómo incorporar el controlador al software, en [Silva, 2012] proponen una forma de integrar un juego con un controlador de un dispositivo móvil utilizando características integradas como retroalimentación de vibración, retroalimentación de sonido, retroalimentación de imagen, reconocimiento de voz, cámara y gestos para mejorar la inmersión del jugador durante el juego. En la arquitectura propuesta, el dispositivo de pantalla móvil se puede usar para dar retroalimentación a los jugadores, como efectos visuales, puntajes y estadísticas. Esta información se pasa a una API (Interfaz de programación de aplicaciones) donde es recopilada para ser analizada. Este esquema ha demostrado ser muy útil y atractivo para los jugadores, brindándoles las opciones que mejor se adaptan a sus preferencias y habilidades. Esta arquitectura también se puede usar en juegos colaborativos, donde cada jugador usa su dispositivo para interactuar con juegos que se ejecutan tanto en dispositivos móviles como en computadoras de escritorio. En nuestra investigación utilizamos un esquema similar para integrar los juegos de computadora con el layout del dispositivo móvil, usando un API con endpoints específicos para cada acción, que ejecutan y recopilan la información de control del usuario.

En [Montenegro, 2018] proponen la implementación de una API que recopila y registra datos de entrada del usuario en el juego. Establecido con estos datos, el sistema reorganiza su diseño de manera dinámica y sin problemas, creando un diseño único y personalizado mejorando la experiencia de usuario. Validan su propuesta con un conjunto

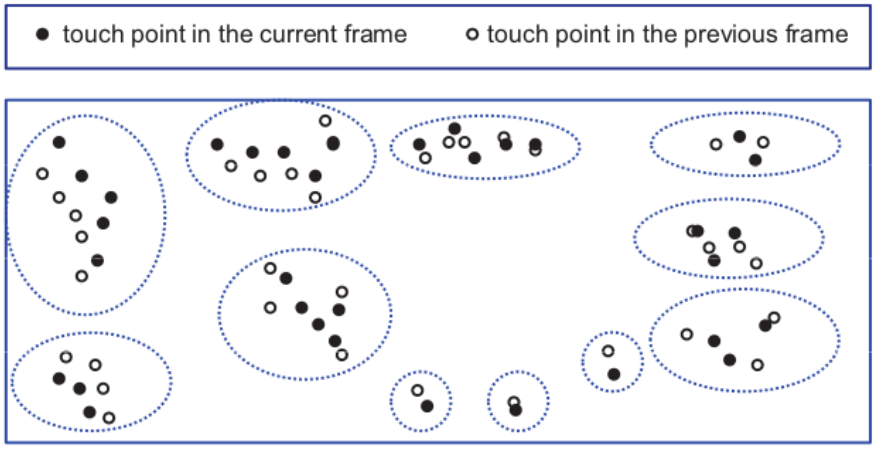

Figura 2.1: Agrupamiento de *touches* en cuadros secuenciales

de jugadores experimentados y no experimentados, además de comparar el controlador virtual propuesto con uno sin modificaciones o adaptabilidad. En esta investigación proponen dos heurísticas para la adaptación de los botones del layout, que mejoran la posición y tamaño de los botones del layout. El primero es de la modificación del tamaño de los botones a partir de la cantidad de veces que han sido utilizados, y la segunda es del reposicionamiento de los botones a partir de los centroides después de someter las posiciones de los toques en el *touchpad* de los usuarios a un algoritmo de agrupamiento llamado k-means. Partimos de muchos conceptos de esta investigación para algunas heurísticas de nuestra propuesta.

Sin embargo, realizar todos los cálculos necesarios para la nueva renderización puede demorar bastante tiempo, mientras que el usuario no puede esperar por todos estos cálculos luego de presionar algún botón. Debido a esto utilizamos un modelo asíncrono. Un modelo asíncrono permite que ocurran varias ejecuciones al mismo tiempo. Trabajamos en un modelo asíncrono para evitar que el cliente espere por todos los cálculos del API para realizar la siguiente renderización y el *layout* dinámico se renderize paulatinamente.

La asincronicidad tambien facilita el escalamiento del API, debido a que podemos utilizar herramientas *serverless* como AWS Lambda o Google Cloud functions para ejecutar tareas independientes. Prosiguiendo por esta idea ejecutar las tareas en grupos, tendria un menor costo computacional, debido a que aprovecharíamos mejor el tiempo de ejecucion por cada funcion, reutilizariamos procesos comunes y podemos adecuar de manera empirica la cantidad de funciones que deben correr en cada funcion. De manera similar se realizo pruebas de escalamiento con kubernetes, utilizando servidores no *serverless* pero que pudieran escalar el numero de nodos basados en el procesamiento promedio del CPU, la ventaja de este escalamiento para el servidor es que se podian procesar grupos de mayor tamaño en cada servidor, reutilizando mas procesamiento previo para cada uno de los servidores.

2.3. Agrupamiento de posiciones de *touches*

El agrupamiento de los puntos de los *touches* es computacionalmente costoso. Empleamos la técnica de agrupamiento geométrico [Huang et al., 2015] para reducir el tamaño del problema y mantener la precisión. El agrupamiento toma los puntos de contacto de dos cuadros secuenciales como su entrada y produce grupos de puntos de contacto en función de la distancia entre cada punto con los anteriores *touches*. La figura 2.1 muestra un ejemplo de agrupamiento en la pantalla. Después del agrupamiento, necesitamos tratar los puntos grupo por grupo de forma independiente para el seguimiento de puntos. En nuestra propuesta damos un mayor detalle de cómo utilizamos este algoritmo,

Capítulo 3

Estado del Arte

Existen diferentes investigaciones centradas en la generación de *layouts*, así como propuestas de métricas y estándares. La mayoría de técnicas dirigidas a la generación de *layouts* están basadas en algoritmos de inteligencia artificial. [Gabriel Alves, 2019] proponen un modelo de generación de *layouts*, que genera múltiples diseños en vez de uno óptimo, agrupando los diseños a través de algoritmos de aprendizaje máquina. Se logró generar diferentes *layouts* o distribuciones del controlador móvil con un alto porcentaje de aceptación, tomando en cuenta características como la posición y la forma. El propósito de esta investigación es generar diferentes diseños para facilitar el espacio de diseño, llegando a implementar una librería o *framework*. Sin embargo, no se tomó en cuenta otros aspectos como la funcionalidad de cada botón o su personalización. Se validó los resultados sometiendolos al criterio de un conjunto de diseñadores profesionales.

La investigación de [Montenegro, 2018] presenta la implementación de un API y un aplicativo móvil que es capaz de personalizar el juego al usuario. La idea es que los touches del usuario sean captados por el aplicativo y categorizados en exitosos o fallidos (dependiendo de si el toque activa algún botón). Esta información es guardada en una base de datos a través del API, y en un thread aparte del aplicativo se agrupan los touches y se calculan los puntos centroides a través de un algoritmo de aprendizaje de máquina. Estos centros serán los puntos donde se deberán colocar los botones. La investigación ofrece la capacidad de que el *layout* pueda cambiar con respecto a la dinámica o etapas del juego, desapareciendo aquellos botones que no son útiles en una etapa inicial, colocando y cambiando los iconos de los elementos del *layout*. La investigación es validada con un conjunto de pruebas, midiendo el desempeño de los jugadores (experimentados y no experimentados) en el juego y su experiencia como usuario.

En la investigación de [Lutteroth, 2017] propone la generación automática del *layout* para el cambio de orientación del dispositivo móvil, haciendo un análisis de aplicativos existentes e identificando patrones comunes entre estos. Basado en esos patrones desarrollaron un prototipo que genera escenarios y *layouts* alternativos de un *layout* inicial. Desarrollando un método de clasificación para reconocer un *layout* de calidad, de las cuales los diseñadores podrán escoger la que mejor se acomode a sus aplicaciones. Validando el estudio con un cuestionario bien estructurado y fácil de realizar.

Con respecto al estudio de métricas, mientras que existe un extenso campo de in-

vestigación en dispositivos de entrada y controladores móviles, no hay una gran cantidad de investigaciones científicos que validen un diseño óptimo para *layouts*. El estudio de [Suette, 2015] se centra en comparar cuatro diferentes disposiciones de *layouts* (botones direccionales, *joystick* flotante, *pad* direccional, control de inclinación) comunes en dos videojuegos clásicos (Pac Man y Super Mario Bros). Su estudio indica la preferencia de usar el *pad* direccional y el *joystick* flotante, mientras que los botones direccionales y el control de inclinación les parece incómodo a los usuarios. El estudio fue validado al realizar un conjunto de pruebas con diferentes jugadores. Esto le permite al paper dar algunas métricas simples para la validación de *layouts* virtuales simples.

En un enfoque similar [Clua, 2017]; presentan un estudio donde buscan comparar los controladores virtuales de un controlador tradicional físico. El objetivo es determinar qué factores deben ser tomados en cuenta o a cuales se les debe dar mayor importancia en un controlador virtual. En el estudio se concluye que ambos controladores son similares en calidad del *layout* y en la experiencia que producen en el usuario, pero que a nivel fisiológico producen diferentes emociones en el usuario. Esta investigación también provee métricas para comparar controladores virtuales. Los resultados fueron validados por un conjunto de usuarios que probaron ambos controladores.

Tomando en cuenta la investigación de [Burak Merdenyan, 2017], la cual busca el modelamiento del controlador a partir de la requerimientos generados de los comentarios o *feedbacks* de varios jugadores, se busca proponer una importancia definida de varios factores para poder saber en qué concentrarse en específico en la generación de un *layout*. Además propone diferentes categorías de evaluación como la duración de la batería, comfort, compatibilidad, facilidad de uso, funcionalidad, calidad del material, reconocimiento, respuesta y manual de usuario; calculando el porcentaje de importancia de cada uno.

Hay otras investigaciones relacionadas a la comparación y validación de los controladores a partir de ciertos criterios del diseño de videojuegos como lo son el confort, el aprendizaje, la sensibilidad y el realismo. Como es explicado por [Pitt, 2010], los cuales establecen una forma de evaluar el uso de los controladores a partir del framework teórico de [Pitt, 2006], la cual es una evaluación rápida y eficiente, que puede ser tomada de punto de partida para comparar diferentes *layouts* y saber cuál es mejor.

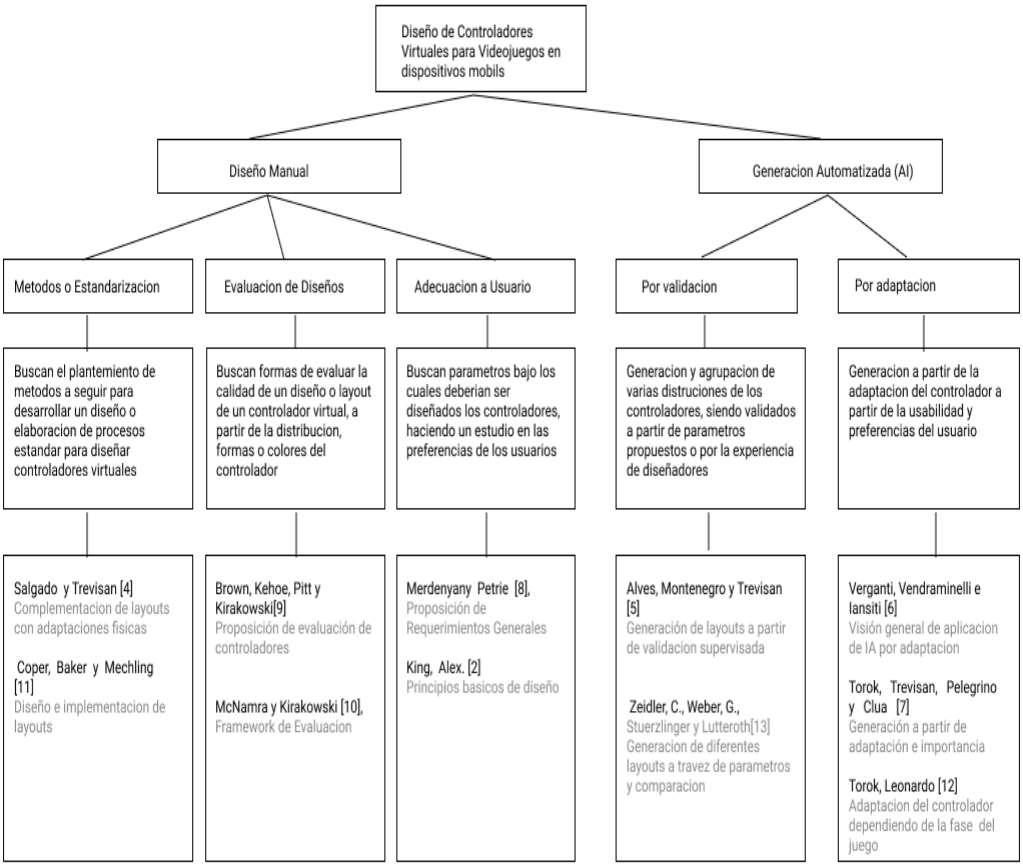

Figura 3.1: Categorización de Papers

Capítulo 4

Propuesta

4.1. Adaptación automática de *layouts* en dispositivos móviles utilizando algoritmos heurísticos y agrupamiento

Para generar un *layout* del dispositivo móvil apropiado al usuario, debemos tomar en cuenta su comportamiento. La forma de captar este comportamiento en la pantalla del dispositivo móvil es a través de los *touches* que realiza el usuario. El método propuesto consta de tres pasos:

1. Registramos las posiciones de los *touches* que realiza el usuario en la pantalla del dispositivo móvil a través de funciones propias del navegador.
2. La información registrada es guardada y analizada a través de diferentes heurísticas en el servidor con el fin de calcular una nueva distribución del *layout*.
3. La información de la nueva renderización es enviada al dispositivo móvil donde se actualiza los componentes como botones o cuadros de texto.

Proponemos la implementación de un *layout* de dispositivo móvil que se adapte dinámicamente al usuario. Ejecutamos el software del controlador en el dispositivo móvil a través de un aplicativo web. Este aplicativo recolecta la información de los *touches* del usuario, y la envía al servidor. Una vez el servidor recibe la información del usuario, agrupa las posiciones de los *touches*, calculando los centroides con el agrupamiento geométrico. Los centroides determinan el color, bordes, tamaño y posición de los nuevos componentes renderizados del *layout*. Por último, el servidor devuelve el análisis realizado al usuario y el aplicativo web vuelve a renderizar el *layout* con la información retroalimentada del servidor.

Se optó por una arquitectura cliente-servidor. El software registra las posiciones de los *touches* del usuario y las envía al servidor o API a través de *sockets*. En nuestro marco teórico mencionamos algunas de las ventajas de utilizar dicho modelo con asincronía y escalamiento, donde sobresale el procesamiento de las tareas en grupos, reutilización de

cálculos y la independencia entre diferentes tareas. Luego el servidor guarda la información de los touches para procesarla en una base de datos, se guarda la información porque puede ser reutilizada en varias iteraciones dependiendo del número de renderizaciones hacia atrás que se consideren para calcular una nueva renderización.

Para lograr la adaptación deseada al estilo de juego de cada usuario es necesario realizar varias adaptaciones. Para este trabajo, se decidió realizar seis diferentes cambios en el diseño del controlador: tamaño, posición, color, transparencia, grosor de bordes y sombras de los botones. Estas adaptaciones son realizadas al mismo tiempo para cada botón, excepto la adaptación de sombras. Están integradas de tal manera que no se contrapongan las adaptaciones entre sí, y tampoco confundan al usuario.

4.2. Heurística de adaptación de tamaño

Utilizamos un enfoque similar al de [Montenegro, 2018] para la adaptación de tamaño, mejorándolo al agregar pesos a cada botón del componente de modo que crezcan más rápido los botones que tienen una implicancia directa en el juego. Un ejemplo directo es que el botón de configuración debería crecer lentamente debido a que tiene un menor uso que uno de movimiento en el Pac Man, la cantidad de veces promedio que es usado un botón equivaldría al peso de ese componente dependiendo de las características del juego. Se puede expresar el nuevo tamaño del botón con la siguiente fórmula:

$$S_b = I_b W_b \frac{\sum_{i=0}^{n} T_{(i,b)}}{n}$$

- S_b: es el nuevo tamaño del botón
- I_b: es el tamaño inicial del botón
- W_b: es el peso del botón, este debe ser asignado antes del cálculo
- n: es la cantidad de touches recientes analizados.
- $T_{(i,b)}$: es un valor que oscila entre 0 y 1, dependiendo de la proximidad de un toque del *touchpad* al botón

Una gran adaptación o radical cambio en el tamaño del botón puede deformar el layout, por lo que se establecen límites en su adaptación, con respecto al tamaño máximo y mínimo del componente:

$$I_b - l \leq S_b \leq I_b + l$$

Donde l es el límite de modificación de tamaño. Si el botón sobrepasa a $I_b + l$, su tamaño se igualara a esta cota. De manera similar si el botón es menor a $I_b - l$ su tamaño no debe reducirse a menos de ese límite.

4.3. Heurística de adaptación de posición

La heurística de adaptación de posición exige un proceso más sofisticado, que intenta encontrar los puntos en la pantalla que representan los centros de las áreas más utilizadas. El algoritmo de agrupamiento presenta una buena solución para este caso, clasificando un conjunto de puntos o toques, en clases que representan los botones. Realizar un cálculo de agrupamiento de las posiciones de los *touches* puede ser un proceso bastante pesado computacionalmente. El agrupamiento más simple como k-means tendría una complejidad práctica de $O(n^2)$. Debido a las características de este problema, decidimos usar el algoritmo de agrupación geométrica [Huang et al., 2015].

Este algoritmo utiliza un algoritmo llamado *sweep line* el cual ordena las posiciones de los puntos con respecto a una dimensión (que en nuestro caso es el eje x) y realiza una barrido de los puntos con un threshold, para agrupar los puntos más cercanos y procesar el clustering en subgrafos. De esta manera simplificamos el cálculo a una complejidad que es prácticamente lineal $O(n)$ según las demostraciones de [Huang et al., 2015], el cual aplica el algoritmo a un screen multitouch, pero en nuestra implementación solo consideramos un único touch por evento. En la investigación se define el problema de agrupamiento de la siguiente forma:

- **Problema de agrupamiento**: dados dos conjuntos P y C de puntos de contacto y un *threshold* T, donde partimos P y C en subgrupos $(P_1, C_1), (P_2, C_2), ..., y(P_k, C_k), 1 \leq k <MAX(m, n)$, tal que el número de subgrupos se maximiza y el seguimiento de subgrupos individuales es finalmente equivalente al seguimiento final sin hacer agrupamiento.

Sin embargo, como solo agrupamos los puntos en una sola dimensión, los *touches* a dos componentes que se encuentren superpuestos o uno encima del otro podrían considerarse como del mismo grupo o componente, por lo que es necesario establecer un radio de distancia para emparejar *touches* que intentaron accionar un botón a partir de los centroides que calculemos. Además para mejorar el algoritmo corrimos otro *sweep line* en el eje y, para que se encuentren mejor agrupados y minimizar el error de emparejamientos en localizaciones cercanas.

Después de encontrar los centroides, cada uno se empareja con el botón más cercano. En casos donde hay dos centroides y el botón más cercano es el mismo, el par correcto será creado en base a la distancia mínima, dejando el otro centroide sin asignar botones. El centroide emparejado con el botón se considerará como su posición óptima, que representa el punto medio de todas las entradas dirigidas a ese botón. Con estos datos, el controlador comenzará a mover el botón gradualmente hacia el centroide de su grupo, hasta que el centro del botón está ubicado precisamente en el centroide correspondiente. Estos cambios en los botones son visuales y el usuario puede observar las adaptaciones que se realizan en tiempo real. En la Figura 4.1, podemos ver una representación del pseudocódigo del algoritmo de agrupamiento geométrico, en el cual se dividen los puntos de conjuntos de *touches* según la renderización en la que hayan sucedido.

En el siguiente ejemplo, agrupamos las posiciones de los *touches* en dos grupos, relacionados por grafos. En la primera intersección del sweep line con los puntos de forma

```
Algorithm: Clustering(P, C, T, N, S)
Input: P /* the set of points of the previous frame */
       C /* the set of points of the current frame */
       T /* threshold */
Output: N /* the number of sub-groups */
        S /* the set of sub-groups */
1 S = ∅
2 N = 0
3 Y = ∅ /* the set of point candidates */
4 Put all points of P and C into U, and
  sort points of U in the non-decreasing x-coordinate order.
5 Put all points into G''
6 Perform line sweeping in U from left to right to add edges for G''
7   if the line meets the point n1 in P
8       Check whether any points in Y and in C
        has the Euclidean distance ≤ T with n1
9       if the points exist
10          add edges between n1 and those points
11  else if the line meets the point n1 in C
12      Check whether any points in Y and in P
        has the Euclidean distance ≤ T with n1
13      if the points exist
14          add edges between n1 and those points
15  Put n1 into Y
16  Delete the points in Y whose x-distances with n1
    are larger than T
17 do
18  Pick one un-traversed point as the root r
19  Traverse G'' from r and record the traversed points as a
    sub-Group L
20  Put L into S
21  N = N + 1
22 until all points in G'' are traversed
23 return N and S
```

Figura 4.1: Algoritmo de agrupamiento geométrico

una arista entre P_1 y C_1, debido a que C_1 se encuentra dentro del intervalo del *threshold*. Luego se realiza un proceso similar con los demás puntos, relacionando cada punto con todos los puntos que se encuentren en el intervalo, formando grafos, y los grafos conexos que queden serán un conjunto.

4.4. Heurística de adaptación de color y transparencia

color de componentes en páginas web. Aplicamos un principio similar a la adaptación de tamaño de los botones, pero en su transparencia y color:

$$F_b = O_b + c(\frac{\sum_{i=0}^{n} T_{(i,b)}}{n})$$

$$E_b = C_b + d(\frac{\sum_{i=0}^{n} T_{(i,b)}}{n})$$

- F_b: es el nuevo valor de transparencia
- I_b: es la transparencia inicial del botón
- c: es una constante para mantener la transparencia entre 0 y 1
- F_b: es una nueva tripleta en RGB de color
- C_b: es la tripleta inicial en RGB del botón

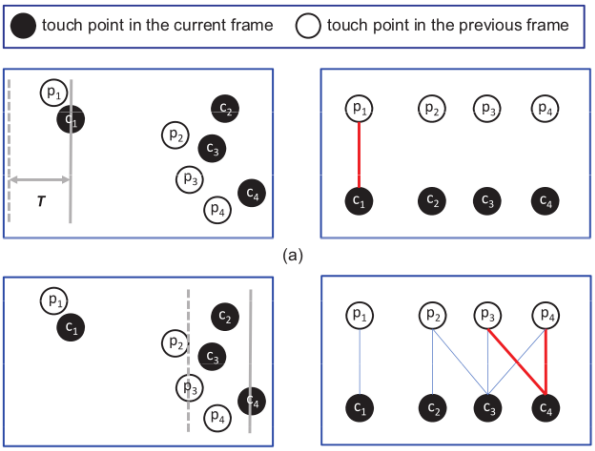

Figura 4.2: Ejemplo de *Sweep Line* con el algoritmo de agrupamiento geométrico

- d: es una constante para mantener el color entre 0 y 255

Si la adaptación en las tripletas fuera en diferentes proporciones, el color cambiara drásticamente, por lo que deben tener una misma variación. Con respecto a la transparencia, el intervalo de variación debe ser pequeño, debido a que ya manejamos un concepto similar en el tamaño del componente, y las dos heurísticas se podrían superponer.

Las adaptaciones en el color y transparencia tienen límites, debido a que cambios bruscos podrían generar confusión y molestia en el usuario. Solo se consideran los últimos n toques en estos cálculos, por lo que el color ayudará a diferenciar los botones más utilizados, simplificando la vista del usuario al buscarlos.

4.5. Heurística de adaptación de grosor de bordes y sombras

Por último con respecto a la adaptación a los bordes, [Koehl, 2012] propone algunas adaptaciones respecto a los bordes de los componentes e imágenes en dispositivos móviles, en nuestro trabajo nos enfocamos en dos aspectos, el grosor y sombra de los bordes de los botones. No podemos cambiar todos los bordes y grosores de los botones por separado, pues si son diferentes, cambiarán la armonía entre los estilos y resultarán confundiendo a los usuarios. El enfoque aquí es cambiar el grosor de los bordes a un mismo número en píxeles. Se puede expresar mediante la siguiente ecuación:

$$B = B_o + y(\frac{\sum_{i=0}^{n} T_{(i,b)}}{n})$$

- B: es el grosor del borde de todos los botones
- B_o: es el grosor del borde inicial de todos los botones

- y: es una constante para mantener el grosor entre 0 y 3 y no reduzca el interior del botón

La adaptación en la modificación del grosor del botón tiene límites, debido a que un grosor muy grande puede deformar el tamaño y aspecto del botón. Respecto a la adaptación de la dirección del botón, se calcula cual es el lado de la pantalla que fue más utilizado y a partir de ese cálculo la sombra es renderizada al lado opuesto, para resaltar el lado que fue más utilizado.

Dado $R_{(i,b)}$ que identifica si el i-ésimo toque fue a la derecha (asignándole un valor del 0 al 1) y $L_{(i,b)}$ que identifica si el i-ésimo toque fue a la izquierda (asignándole un valor del 0 al 1); si se cumple que $\sum_{i=0}^{n} R_{(i,b)} < \sum_{i=0}^{n} L_{(i,b)}$ entonces la mayoría de toques se dieron a la izquierda, por lo que la sombra de los botones se debe dirigir a la derecha, caso contrario se debe dirigir a la izquierda.

Capítulo 5

Experimentación y Resultados

5.1. Implementación

El servidor fue implementado en Django debido a la flexibilidad que este framework ofrece. Como consecuencia de que el controlador virtual tiene repercusiones en tiempo real sobre el lado desktop del sistema, se usó websockets para poder actualizar el estado del juego simultáneamente con la actualización del *layout* del controlador. Se utilizó el framework Django Channels para poder interactuar con los websockets, así como el framework Django REST para algunas interacciones que no necesitan de una comunicación bidireccional entre el cliente y el servidor. La definición de tablas de la base de datos MySQL, así como el modelo entidad relación, fue definida a través de modelos y migraciones definidos en Django.

En el servidor también se programaron las secuencias y servicios para realizar el análisis de los touches y procesarlos para hallar los botones más usados así como los centroides. Se consideraron solo los últimos touches, cuyo número también es considerado como parámetro de estudio en los experimentos. Con respecto a la arquitectura del cliente, fue realizada con Django templates, y complementada con javascript, HTML y CSS. El juego desarrollado es Pacman, por lo que se usó la herramienta modernizr de javascript para implementar el juego. Con respecto al controlador virtual, se captaron los touches por medio de funcionalidades ya presentes en HTML5. La vista del cliente es accedida por medio de los endpoints del servidor. Otro parámetro considerado en los experimentos es el límite de movimiento de un botón, esto para evitar la deformación del controlador.

Para relacionar el controlador del dispositivo móvil con el juego en *desktop* utilizamos un token para identificar al usuario, y saber con qué dispositivo conectar las acciones del controlador del usuario. Se puede apreciar en la Figura 5.1 el controlador móvil y el juego probado.

Figura 5.1: *Layout* simple con el que se realizó la experimentación

5.2. Despliegue

Con respecto al despliegue del servidor, se utilizó en un comienzo AWS Lambda, debido al bajo costo y nivel de concurrencia. Se llevaron a cabo varias pruebas y existía una fricción con Django channels en sesiones del usuario muy largas. Por lo que decidimos utilizar EC2 simples en un ambiente básico de Kubernetes en EKS, donde utilizamos un servidor para que procesa las peticiones por Django REST, y un servidor para que procesa las peticiones con Django Channels, ambos sincronizados por el uso de la misma base de datos. Además sólo consideramos un *breadcrumb*, y bloqueamos aquellos que no estuvieran en posición horizontal, para simplificar la experimentación.

La asincronicidad de las tareas fue muy importante para procesar la gran cantidad de *request* que se envían al servidor. Procesar request por request es demasiado caro computacionalmente, por lo que agrupamos los requests, realizamos cálculos comunes y procesamos los requests por grupos, de modo que actualizaban la pantalla del cliente en algunos segundos, pero se mantiene la calidad de renderización (que tiene que ver mucho con la cantidad de *frames* por segundo), por lo que el usuario no se veía afectado por la gran cantidad de cálculos.

5.3. Experimentación

Con el fin de validar cómo se comporta el *layout* de adaptación propuesto con el usuario final y su experiencia de juego, realizamos una prueba de usabilidad, observando la efectividad y satisfacción del usuario con respecto a nuestras adaptaciones. Para esta evaluación, la dividimos en dos etapas: las pruebas de configuración y las pruebas de usuarios. La prueba de configuración se utilizó para definir parámetros de adaptación importantes en las pruebas con los usuarios, determinando un configuraciones de calidad para la adaptación.

La evaluación utilizó dos controladores diferentes, uno adaptativo basado en nuestra

propuesta, otro adaptativo basado en otra investigación [Montenegro, 2018] y otro no adaptativo, todos con la misma funcionalidad y diseño. Se probó ambos controladores con los usuarios, tomándose tres evaluaciones de sesiones por usuario. Nuestra comparación se centró en comparar el controlador adaptativo propuesto con una versión estática del layout y también con la propuesta de otro layout dinámico basado solo en las heurísticas de tamaño y posición.

Sin embargo, es difícil determinar si una interfaz es óptima, ya que requiere varias pruebas de uso y validación constante. Este puede ser costoso y poco práctico debido a que exige un tiempo que no podría estar disponible para las personas que desarrollan el layout, por lo que usamos 3 métricas de validación directas para comparar los métodos. Con estas pruebas de usabilidad esperamos mostrar cómo un controlador adaptativo puede mejorar un controlador de juego personalizado y mejorarlo a una configuración más cómoda o útil para el usuario.

Nuestro grupo de voluntarios estuvo formado por 20 usuarios, 9 hombres y 11 mujeres, con edades que van desde los 20 a no más de 30 años. El grupo de usuarios fue seleccionado y separado en dos grupos diferentes: 10 usuarios expertos y 10 usuarios poco experimentados. El primer grupo fue compuesto por usuarios acostumbrados a juegos en dispositivos móviles con regularidad. El segundo grupo fue creado por usuarios con menos experiencia con los juegos y en su mayoría consumidores de juegos sencillos.

El servidor nodo que realiza los cálculos de renderización de los controladores se ejecutó en dos instancia EC2 t2 micro (por nodo), por su capacidad de procesamiento se evitó alta concurrencia que las pruebas se dieran de manera simultánea para no ocupar la RAM de esta instancia de AWS (Lo que podría ralentizar). Idealmente se debería usar un middleware como Apache Kafka para guardar los requests en colas. Los celulares utilizados por los usuarios no tenían alguna marca o sistema operativo en especial, pero sus tamaños no superan los 450 x 800 píxeles. El juego con el cual se probará nuestra propuesta es Pac-Man, debido a que la simpleza del juego no interferirá con las métricas que se desean evaluar.

El controlador adaptativo debe configurarse con dos parámetros que afectan el proceso de adaptación: el límite de movimiento del botón por iteración, medido en píxeles y la cantidad de los puntos más recientes enviados al algoritmo de agrupamiento geométrico. Para el primero, un valor más alto hará que el controlador cambie su diseño más rápido. En el segundo parámetro, un valor menor hará que el controlador considere solo puntos recientes, adaptándose más rápidamente a las condiciones actuales de la interacción.

5.4. Métricas

Para la evaluación de nuestra propuesta evaluaremos tres métricas que consideran la precisión del layout, su utilidad y la comodidad del usuario [Suette, 2015]:

- *Touch Success*: Calcula la razón entre el número de *touches* con respecto al total de *touches*, esta métrica podría aumentar o disminuir de acuerdo al radio de aceptación

Setup	Success rate
5 pixeles, 80 points	92.63
10 pixeles, 50 points	92.80
50 pixeles, 30 points	94.75
100 pixeles, 10 points	93.93

Tabla 5.1: Resultados de pruebas de configuración

del *touch*, que sirve para delimitar si un usuario intentó accionar cierto componente. Está muy relacionado con la precisión del *layout*.

- *Task Success*: Calcula la razón de tareas no completadas por un usuario con respecto al total de tareas existentes, esta métrica podría aumentar o disminuir de acuerdo a la complejidad del juego. Está muy relacionado con la utilidad del *layout*.

- *Subject Score*: Es el puntaje que el usuario asigna al layout. Está muy relacionado con la comodidad del usuario.

5.5. Resultados

Los resultados del experimento mostraron que la adaptación más rápida le permitió al controlador reaccionar rápidamente a los cambios en el estilo de juego del usuario, como situaciones en las que el usuario cambió la posición de sus manos sin darse cuenta. En una evaluación subjetiva, la adaptación más rápida proporciona un controlador que se adapta mejor al juego y tiene una respuesta rápida para corregir los errores del usuario. Si la adaptación fuera demasiado rápida, también podría confundir al usuario, por lo que es importante hacer esta configuración inicial. Al probar esta configuración, utilizamos la tercera y cuarta configuración (movimiento de píxeles, crecimiento de tamaño), para estandarizar los resultados en todos los usuarios. Las primeras dos configuraciones tenían adaptaciones muy lentas como para ser notadas por el usuario y menor cantidad de *touches* exitosos (*Success Rate*), por lo que no fueron considerados en la experimentación. Podemos ver estos resultados en la Tabla 5.1.

Una vez definidos los parámetros de adaptación iniciamos las pruebas de usabilidad con usuarios. Cada sesión de evaluación se limitó a seis minutos de juego y aproximadamente dos minutos de entrenamiento antes de comenzar la prueba. La sesión de entrenamiento consistió en una interacción libre del usuario mientras se le explicaba la función de cada botón, así como los objetivos del juego. La evaluación comprende tanto una encuesta subjetiva recopilada por un cuestionario y una investigación objetiva registrada por los datos de registro recopilados durante las interacciones del usuario.

La evaluación subjetiva incluye problemas de rendimiento y usabilidad percibidos. Todos los eventos causados por la interacción del dispositivo de control de usuario se escribieron en un registro de la base de datos, datos tales como el registro de los toques, la tasa de éxito del controlador para cada período de tiempo, la tasa de éxito de cada botón, la puntuación y la cantidad de vidas gastadas en el juego. La tasa de éxito es un valor entre 0 y 100, que representa el porcentaje de toques correctos. La tasa de éxito por

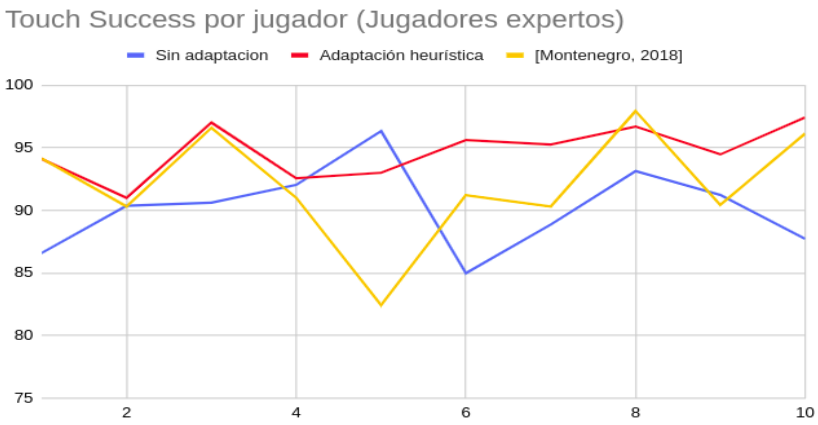

Figura 5.2: *Touch Success* de los jugadores expertos

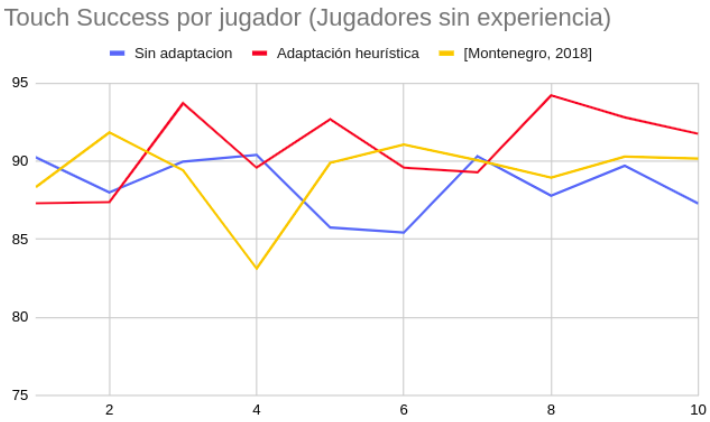

Figura 5.3: *Touch Success* de los jugadores sin experiencia

botones se calcula de manera similar, pero solo teniendo en cuenta los toques destinados a ese botón específico. Cada toque correcto está asociado al botón que el usuario presionó y cada toque incorrecto es asociado con el botón más cercano.

En la Figura 5.2 se muestra una comparación del desempeño de los 10 jugadores expertos respecto a la métrica *touch success*. El *layout* sin adaptación tiene un promedio de 90,19, mientras que la adaptación heurística (nuestra propuesta) tiene un promedio de 94,72 y la de [Montenegro, 2018] tiene 92,05. Nuestra implementación dio resultados ligeramente mejores que el de [Montenegro, 2018], porque la adaptación en los bordes permite al usuario diferenciar más fácilmente las áreas de los botones, evitando que se equivoquen al momento de presionarlos.

En la Figura 5.3 se muestra una comparación del desempeño de los 10 jugadores inexpertos respecto a la métrica *touch success*. El *layout* sin adaptación tiene un promedio de 88,49, mientras que la adaptación heurística tiene un promedio de 90,83 y la de [Montenegro, 2018] tiene 89,31. No se muestra una gran diferenciación entre el *layout* adaptado por nuestro método y los otros comparados.

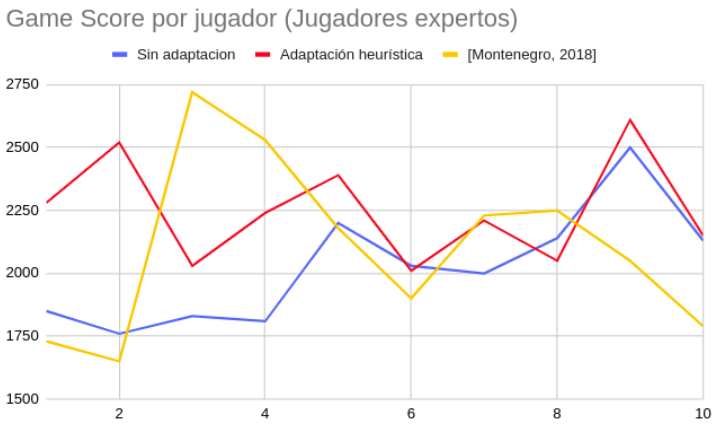

Figura 5.4: Game Score de los jugadores expertos

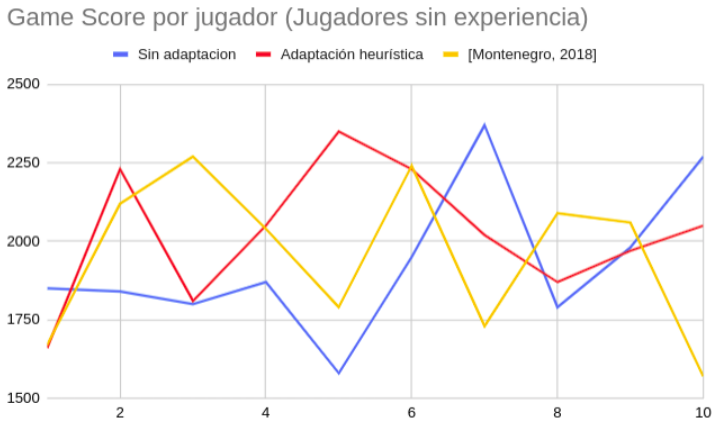

Figura 5.5: *Game Score* de los jugadores sin experiencia

En la Figura 5.4 se muestra una comparación del desempeño de los 10 jugadores expertos respecto a la métrica *game score*. El *layout* sin adaptación tiene un promedio de 2025, mientras que la adaptación heurística tiene un promedio de 2249 y la de [Montenegro, 2018] tiene 2103. Nuestra implementación dio resultados ligeramente mejores, porque los parámetros de adaptación hicieron que la adaptación sea más rápida, además de que la historia de adaptación y peso de los touches hacen que la adaptación sea más precisa frente a jugadores expertos.

En la Figura 5.5 se muestra una comparación del desempeño de los 10 jugadores inexpertos respecto a la métrica *game score*. El *layout* sin adaptación tiene un promedio de 1930, mientras que la adaptación heurística (nuestra propuesta) tiene un promedio de 2024 y la de [Montenegro, 2018] tiene 1958. La implementación de nuestra propuesta no superó a la de [Montenegro, 2018] y no se muestra una gran diferenciación con el *layout* sin adaptación. Probablemente nuestras modificaciones no tengan un gran efecto en el desempeño de jugadores inexpertos.

En la Figura 5.6 se muestra una comparación de la encuesta a los 10 jugadores

Figura 5.6: Puntaje subjetivo de jugadores expertos

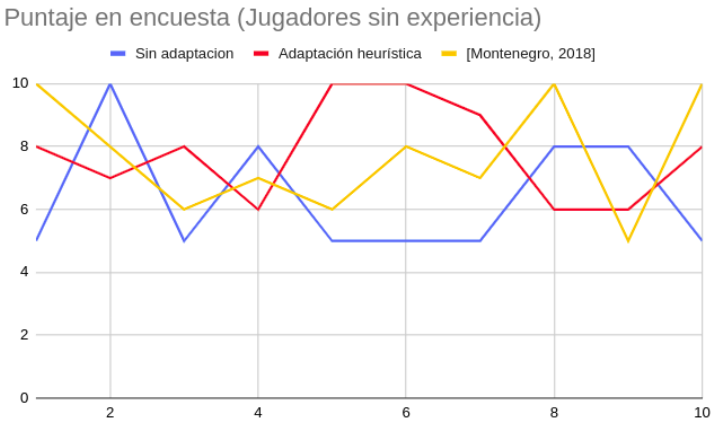

Figura 5.7: Puntaje subjetivo de jugadores sin experincia

expertos respecto a cómo se sintieron con los *layouts*. El *layout* sin adaptación tiene un promedio de 7,9, mientras que la adaptación heurística (nuestra propuesta) tiene un promedio de 7,9 y la de [Montenegro, 2018] tiene 7,1. La propuesta de [Montenegro, 2018] resultó siendo la menos atrayente . Esto los atribuimos a los diferentes cambios del layout que llamaron la atención de los jugadores.

En la Figura 5.7 se muestra una comparación de la encuesta a los 10 jugadores sin experiencia respecto a cómo se sintieron con los *layouts*. El *layout* sin adaptación tiene un promedio de 6,4, mientras que la adaptación heurística (nuestra propuesta) tiene un promedio de 7,8 y la de [Montenegro, 2018] tiene 7,7. Al puntuar los *layouts* los jugadores puntuaron mejor a la implementación de nuestra propuesta. Esto los atribuimos a los diferentes cambios del layout que llamaron la atención de los jugadores.

5.6. Análisis de los resultados

Como pudimos notar, el que se aplique cualquiera de las adaptaciones sobre un layout mejora de por sí la experiencia del usuario (en cualquiera de las tres métricas). Las mejoras que se hicieron sobre el color, transparencia, bordes y sombras realizaron una mejora sobre las métricas de *touch success* y *Subject Score*; debido a que estas adaptaciones se centran en mejorar la diferenciación de los botones con respecto al entorno del layout y de los demás botones. Las adaptaciones de posicionamiento y tamaño estaban presentes tanto en la implementación de nuestra propuesta como la de la investigación comparada [Montenegro, 2018]. La adaptación dinámica ayudó a evitar que el usuario realizará toques incorrectos en la pantalla del dispositivo móvil. Probablemente en una muestra más grande se conseguirían mejores resultados para comparar ambas adaptaciones.

Capítulo 6

Conclusiones

En este trabajo implementamos un controlador virtual para dispositivos móviles capaz de adaptarse al comportamiento del usuario a partir de la recopilación y análisis de los *touches* de un usuario. Nuestra implementación adapta dinámicamente el tamaño, forma, color, transparencia, grosor de borde y sombras, a partir de heurísticas. Como analizamos en nuestros resultados, aplicar esta adaptación mejora la experiencia y desempeño del usuario.

Nuestro trabajo logró una mejora en la creación de *layouts* adaptables al usuario, además que los usuarios acertarán mayor número de veces a los componentes del controlador así como que realizarán las tareas del juego con mayor puntaje. Notamos que aplicar cualquier adaptación da mejores resultados que no aplicar ninguna, por lo que el diseño de *layouts* dinámicos en los controladores móviles o virtuales es un tema que merece más investigación para mejorar la experiencia del usuario. Debemos buscar hacer a los controladores más inteligentes e interactivos con los usuarios.

Sin embargo no basta con agregar múltiples funcionalidades de adaptación debido a que se pueden superponer o hasta complicar la lógica del juego. Mantener un controlador intuitivo y simple siempre debe ser una prioridad al momento de crearlos. Por lo que siempre se deben manejarse límites de adaptación o implementar alguna forma de controlar la lógica o consistencia de todo el *layout*.

Si bien logramos reducir el procesamiento por el agrupamiento, la cantidad de cálculos que son necesarios por renderización sigue siendo alto, por lo que debemos reducir el número de peticiones al servidor calculando el número de errores o *touches* mal direccionados en el dispositivo del usuario. No tiene sentido cambiar el diseño del *layout* si el controlador tiene una alta eficiencia, por lo que debería reducir el costo computacional de nuestra propuesta.

Como trabajo futuro, las técnicas revisadas deberían centrarse en controladores más generales. Por ejemplo la distribución de teclados virtuales. También debemos considerar las cualidades semánticas de los componentes del diseño, como la relación de un color/texto con la funcionalidad del componente (semántico, funcional), una característica importante en la calidad del juego. Ejecutar en paralelo las tareas asíncronas es otra forma de reducir este tiempo de ejecución.

Bibliografía

[Bi, 2013] Bi, X., Z. S. B. (2013). A statistical criterion of target selection with finger touch. *ACM UIST '13*.

[Burak Merdenyan, 2017] Burak Merdenyan, H. P. (2017). User reviews of gamepad controllers: A source of user requirements and user experience. *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*.

[Clua, 2017] Clua, G. G. . E. M. . L. T. . D. T. . A. M. . E. (2017). Understanding user experience with game controllers: A case study with an adaptive smart controller and a traditional gamepad. *16th International Conference on Entertainment Computing*.

[Gabriel Alves, 2019] Gabriel Alves, Anselmo Montenegro, D. T. (2019). Creating layouts for virtual game controllers using generative design. *Joint International Conference on Entertainment Computing and Serious Games*.

[Huang et al., 2015] Huang, S.-L., Hung, S.-Y., and Chen, C.-P. (2015). Clustering-based multi-touch algorithm framework for the tracking problem with a large number of points. pages 719–724.

[Koehl, 2012] Koehl, Aaron Wang, H. (2012). Efficient content adaptation for mobile devices. *Middleware 2012*.

[Lutteroth, 2017] Lutteroth, C. Z. . G. W. . W. S. . C. (2017). Automatic generation of user interface layouts for alternative screen orientations. *IFIP Conference on Human-Computer Interaction*.

[Montenegro, 2018] Montenegro, L. T. . D. T. . M. P. . A. (2018). Smart controller: Introducing a dynamic interface adapted to the gameplay. *Entertainment Computing 27*.

[Pitt, 2006] Pitt, M. B. . A. K. . J. K. . I. (2006). Functionality, usability, and user experience: three areas of concern. *Interactions ACM*.

[Pitt, 2010] Pitt, M. B. . A. K. . J. K. . I. (2010). Beyond the gamepad: Hci and game controller design and evaluation. *Evaluating User Experience in Games (pp.197-219)*.

[Silva, 2012] Silva, M. J. . R. D. (2012). An architecture for game interaction using mobile. *2012 IEEE International Games Innovation Conference*.

[Stewart, 2013] Stewart, R. S. . J. . (2013). Fingercloud: uncertainty and autonomy handover in capacitive sensing. *ACM CHI '13*.

[Suette, 2015] Suette, M. B. . P. F. . F. A. . S. (2015). Investigating on-screen gamepad designs for smartphone-controlled video games. *ACM Transactions on Multimedia Computing Communications and Applications*.

[Weir, 2012] Weir, D., R. S. M. S. R. L. (2012). A user specific machine learning approach for improving touch accuracy on mobile devices. *ACM UIST '12*.

[Zamith, 2013] Zamith, M., J. M. S. J. (2013). Adaptcontrol: An adaptive mobile touch control for games. *SBGames, 137-145*.