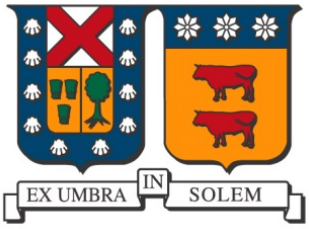
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“MITIGANDO ERRORES DE CONCEPTO EN ESTUDIANTES DE CURSOS INTRODUCTORIOS DE PROGRAMACIÓN A TRAVÉS DE UN VIDEOJUEGO”

DANIEL FRANCISCO ALVAYAY FIGUEROA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Federico Meza
Profesora Correferente: Andrea Vásquez

Julio - 2024

DEDICATORIA

A Eugenia, Leonila, Baltazar, Ron y Toby, por acompañarme durante este camino.

AGRADECIMIENTOS

Quiero agradecer principalmente a mi familia por el apoyo que me han dado. A mi mamá, por estar siempre presente y por todo lo que ha hecho por mí. A mi abuelita, por haberme cuidado y darme herramientas que me han servido hasta el día de hoy. A Baltazar, por haberme acompañado tantos años y alegrar mis días. Y a Ron y Toby, por la felicidad que me han dado en este tiempo.

También quisiera agradecer al profesor Federico y al paralelo con el que pude validar la propuesta, por haberme dado una nueva mirada sobre cómo potenciar la educación con algo que siempre me ha gustado, los juegos.

Por último, agradezco a mis amigos y compañeros de universidad, por los buenos momentos que pasé junto a ellos desde que ingresé a la carrera.

RESUMEN

Resumen— En este trabajo se presenta el diseño, desarrollo y validación de *Pyconception*, un videojuego hecho para ayudar a disminuir errores de concepto y fortalecer el desarrollo del pensamiento computacional en un curso universitario de programación para estudiantes de primer año. El juego consiste en recoger llaves de colores vinculadas a variables de código en *Python*, presentando actividades de baja carga cognitiva que involucran errores intencionales que los jugadores deben detectar para progresar. Se llevaron a cabo 5 sesiones de prueba, cada una con una duración de 30 minutos.

La propuesta se validó mediante pruebas basadas en inventarios de concepto aplicadas a dos grupos, uno de los cuales había jugado previamente. Los resultados del grupo experimental fueron más altos y consistentes que los obtenidos por el grupo de control, demostrando que *Pyconception* logró reducir los errores de concepto. Adicionalmente, el juego fue bien recibido por el estudiantado, quienes lo valoraron como una herramienta valiosa para estudiantes que cursan una asignatura de programación por primera vez.

Palabras Clave— *misconceptions*; Pensamiento computacional; programación; videojuego.

ABSTRACT

Abstract— This work presents the design, development, and validation of *Pyconception*, a video game created to help reduce conceptual errors and strengthen the development of computational thinking in a university programming course for first-year students. The game involves collecting color-coded keys linked to code variables in *Python*, presenting low-cognitive-load activities that include intentional errors that players must identify to progress. Five trial sessions were conducted, each lasting 30 minutes.

The proposal was validated through tests based on concept inventories applied to two groups, one of which had played the game previously. The results of the experimental group were higher and more consistent than those of the control group, demonstrating that *Pyconception* effectively reduced conceptual errors. Additionally, the game was well-received by the students, who valued it as a valuable tool for those taking a programming course for the first time.

Keywords— *misconceptions*; Computational thinking; programing; videogame.

GLOSARIO

DI: Departamento de Informática.
IA: Inteligencia Artificial.
IoT: Internet of Things (Internet de las cosas).
MINEDUC: Ministerio de Educación.
PAES: Prueba de Acceso a la Educación Superior.
RPG: *Role Playing Game* (Juego de rol).
STEM: *Science*, *Technology*, *Engineering and Math* (Ciencias, Tecnología, Ingeniería y Matemáticas).
TI: Tecnologías de la Información.
UI: User Interface (Interfaz de usuario).
UTFSM: Universidad Técnica Federico Santa María.
UVA: Unidad Virtual de Aprendizaje.

ÍNDICE DE CONTENIDOS

RESUMEN iv

ABSTRACT iv

GLOSARIO v

ÍNDICE DE FIGURAS viii

ÍNDICE DE TABLAS viii

INTRODUCCIÓN 1

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA 3
1.1 Pensamiento computacional, programación y errores de concepto 3
1.2 Situación a nivel nacional 4
1.2.1 Experiencia previa 4
1.2.2 Conocimiento en otras disciplinas 4
1.2.3 Autoeficacia 5
1.2.4 Motivación 5
1.3 Videojuegos en el aprendizaje de la programación 6
1.4 Objetivos de la Memoria 7
1.4.1 Objetivo general 7
1.4.2 Objetivos específicos 7
1.5 Alcances de la Memoria 7

CAPÍTULO 2: MARCO CONCEPTUAL 8
2.1 Importancia del pensamiento computacional 8
2.2 Errores de concepto 9
2.3 Videojuegos como medio didáctico 10
2.3.1 Aportes de los videojuegos 11
2.3.2 Desafíos de los videojuegos 11
2.3.3 Ejemplos de videojuegos educativos 12

CAPÍTULO 3: PROPUESTA DE SOLUCIÓN 16
3.1 Presentando *Pyconception* 16
3.2 Minijuegos 19
3.3 Alcance de *Pyconception* 22
3.4 Decisiones de diseño 22
3.4.1 Motor del juego 22
3.4.2 Sonidos en el juego 23
3.4.3 Interacción del jugador con el juego 23
3.4.4 *Logs* 25
3.4.5 Primera versión de *UI* 26

3.4.6 Paneles tutoriales . . . 27

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN **29**
4.1 Descripción del proceso . . . 29
4.1.1 *Hardware* utilizado en ayudantías . . . 30
4.2 Análisis cuantitativo de resultados . . . 30
4.2.1 UVA 2: Secuenciales . . . 31
4.2.2 UVA 3: Condicionales . . . 32
4.2.3 UVA 4: Ciclos . . . 33
4.2.4 UVA 5: *Strings* . . . 34
4.2.5 UVA 6: Funciones . . . 35
4.2.6 Desempeño de los grupos . . . 36
4.3 Test cualitativo . . . 36
4.4 Observaciones de las experiencias . . . 38

CAPÍTULO 5: CONCLUSIONES **39**
5.1 Gestión y desarrollo del videojuego . . . 39
5.2 Determinar *misconceptions* . . . 40
5.2.1 Incorporación de *misconceptions* al juego . . . 40
5.2.2 Validación de la efectividad de *Pyconception* . . . 41
5.3 Impacto de *Pyconception* en los estudiantes . . . 42
5.4 Trabajo futuro . . . 42
5.4.1 Nuevos minijuegos . . . 43
5.4.2 Abordar más unidades . . . 43
5.4.3 Enfoque cooperativo . . . 43

ANEXOS **44**
A. Post test . . . 44
B. Test cualitativo . . . 52

REFERENCIAS BIBLIOGRÁFICAS **53**

ÍNDICE DE FIGURAS

1 Ejercicio Bebras de ejemplo para niños de 5 a 8 años. 9
2 Ejercicio Bebras de ejemplo para jóvenes de 17 a 18 años. 9
3 Imagen de uno de los mapas de *Code Combat*. 12
4 Imagen de un nivel de *Code Combat*. 12
5 Una actividad de los videojuego de la Hora del Código. 13
6 Otra actividad de los videojuego de la Hora del Código. 13
7 Nivel 1 del videojuego GIPPPY. 14
8 Nivel 9 del videojuego GIPPPY. 14
9 El *ninja* que controla el jugador, los tipos de llaves y la puerta de meta. 16
10 Ejemplo de código principal. 17
11 Panel de apuntes y ejemplos de uso. 17
12 Controles de movimiento del *ninja*. 18
13 Ejemplos de uso del panel de inventario. 18
14 Controles para el *zoom* de la cámara. 18
15 Todos los tipos de letreros y activación de minijuegos. 19
16 Primer ejemplo de un minijuego. 19
17 Animación de victoria y derrota al terminar un minijuego. 20
18 Segundo ejemplo de un minijuego. 20
19 Tercer ejemplo de un minijuego. 21
20 Menú principal. 23
21 Menú de selección de niveles. 24
22 Panel de botones. 24
23 Menú de pausa. 25
24 Ejemplo de un *log* del nivel 1. 25
25 Primera versión de *UI* para *Pyconception*. 26
26 Última versión de *UI* para *Pyconception*. 27
27 Ejemplo de tutorial. 28
28 Resultados de ambos grupos en el post test 1. 31
29 Resultados de ambos grupos en el post test 2. 32
30 Resultados de ambos grupos en el post test 3. 33
31 Resultados de ambos grupos en el post test 4. 34
32 Resultados de ambos grupos en el post test 5. 35
33 Líneas de tendencia de ambos grupos en los post test. 36

ÍNDICE DE TABLAS

1 Estudiantes que terminaron el taller del MINEDUC en 2017. 6
2 Resultados del test de pensamiento computacional. 29
3 Respuestas al test cualitativo del grupo experimental. 37
4 Respuestas al test cualitativo del grupo de control. 37

INTRODUCCIÓN

Cada año, **el uso de las TI**[1] **ha ido en aumento** y cada vez más personas las utilizan para sus actividades cotidianas. Esto se debe a que pueden facilitar procesos que, de otro modo, requerirían de un gran esfuerzo, como el acceso a información en varios idiomas, la comunicación entre personas de diferentes zonas geográficas, la realización de transacciones bancarias en línea, etc.

Este avance masivo se sitúa en el marco de la **transformación digital**, en el cual las empresas buscan llevar sus procesos a los medios digitales y modernizarlos, con el fin de optimizar sus recursos y obtener una ventaja competitiva. Si bien hoy en día se cuenta con herramientas muy útiles como la IA[2] o el *IoT*[3], **los ingenieros informáticos siguen siendo fundamentales en este progreso**, pues son son capaces de dar forma a soluciones innovadoras para algún problema en específico.

Un conjunto de habilidades clave a la hora de programar forman parte de lo que se conoce como **pensamiento computacional**, ya que permite identificar los aspectos más importantes de un problema, reconocer patrones y abstraerlos para posteriormente crear un algoritmo como solución. **Este conjunto de habilidades se desarrolla comúnmente durante cursos introductorios de programación**, los cuales, en su mayoría, se dictan en instituciones de educación superior; complementándolas con el aprendizaje de algún lenguaje de programación. Sin embargo, los cursos de programación en su formato tradicional cuentan con una **curva de aprendizaje inicial alta**.

Esta curva de aprendizaje puede verse verse negativamente afectada por la aparición de diferentes **errores de concepto** [Qian y Lehman, 2017], ocasionando que los estudiantes se estanquen al percibir la programación como una actividad difícil y desmotivante. En casos extremos, esto podría incluso conducir a la deserción académica [Kazimoglu, 2020].

Para ayudar a disminuir la curva de aprendizaje, se han desarrollado varias alternativas al sistema de clases tradicional, como es el caso de los **videojuegos**, con los cuales se puede reforzar lo aprendido durante las clases al **poner en práctica las habilidades de pensamiento computacional y programación en un entorno interactivo**. Se han realizado múltiples aportes en este aspecto, ya sea a pequeña o a gran escala, lo que ha logrado situar a los videojuegos como una potente herramienta pedagógica que es capaz de motivar a los estudiantes y ayudarlos a mejorar sus habilidades y su desempeño académico [CodeCombat, 2013] [Kodea, sf] [Lara Mira, 2022].

[1] Tecnologías de la información.
[2] Inteligencia Artificial.
[3] *Internet of Things* (Internet de las Cosas).

La propuesta de esta Memoria consiste en desarrollar un videojuego que presente varios *misconceptions* durante sus niveles para que apoye el desarrollo de habilidades de pensamiento computacional y mitigue errores de concepto.

Para validar la propuesta, se espera que los potenciales usuarios sean **estudiantes de primer año que se encuentren cursando la asignatura de Programación**, impartida por el DI[4] de la UTFSM[5], ya que es una asignatura que se debe cursar en la mayoría de las carreras ofrecidas por esta institución; siendo para la mayoría de los estudiantes, la primera vez que aprenden a programar. Además, como gran parte de esos estudiantes se encuentran en una transición desde la enseñanza media hacia la educación superior y deben adaptarse a un nuevo entorno, conocer nuevas personas y cambiar sus hábitos de estudio para enfrentar las nuevas asignaturas; este videojuego puede ser de ayuda y facilitar el proceso de aprendizaje.

Para el desarrollo de la propuesta, se cuenta con una metodología que contempla una etapa de **investigación de los errores de concepto más comunes** en la asignatura de Programación, otra etapa para el **diseño de niveles**, posteriormente el **desarrollo del videojuego** y **sesiones de prueba con los usuarios**. Paralelamente, se dejará registro en este documento el trabajo realizado, finalizando con el análisis pertinente de los resultados obtenidos.

[4]Departamento de Informática.
[5]Universidad Técnica Federico Santa María.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

1.1. Pensamiento computacional, programación y errores de concepto

Una competencia crucial para los profesionales, especialmente para aquellos en el campo de la informática, es contar un alto nivel de **pensamiento computacional**. Este tipo de pensamiento analítico busca resolver problemas de manera estructurada y lógica mediante la computación. Wing lo define como "un enfoque para resolver problemas, diseñar sistemas y comprender el comportamiento humano que se basa en conceptos fundamentales de la computación"[Wing, 2008], destacándolo como una habilidad de suma importancia.

El desarrollo del pensamiento computacional se encuentra ligado (pero no limitado) a la **programación**, cuyos cursos introductorios son comunes en la mayoría de las carreras relacionadas a la ingeniería, especialmente la informática. En estos cursos, se enseña a pensar computacionalmente de forma implícita, fomentándolo a través de los siguientes aspectos:

- Identificación del problema.
- Reconocimiento de patrones.
- Abstracción.
- Descomposición del problema en subproblemas más sencillos de resolver.
- Pensamiento algorítmico.
- Codificación de la solución, ya sea en pseudocódigo o en un lenguaje de programación.

Aprender a programar es una tarea difícil, debido al gran **esfuerzo mental requerido por una persona para realizar una tarea, también conocido como carga cognitiva** [Sweller *et al.*, 1998]. Esto ocurre porque a menudo se requiere realizar varias de las tareas anteriormente descritas de forma simultánea.

Durante el proceso de aprendizaje, es común que aparezcan ***misconceptions*** o **errores de concepto** [Sorva, 2012]. Estos corresponden a **ideas equívocas** que las personas tienen acerca de un tema, en este caso, la programación. Suelen ocurrir al mezclar conceptos (como por ejemplo, el de variable en matemáticas y en un lenguaje de programación) o al no entenderlos bien. Entre los errores más comunes cometidos por los estudiantes de programación, se encuentran los de tipo conceptual, de asignación de variables, secuencias de control (tales como el manejo de condicionales o ciclos), funciones, etc [Vásquez Pinto, 2020].

1.2. Situación a nivel nacional

Actualmente en Chile, **los cursos de programación se imparten en su mayoría en instituciones de educación superior**, específicamente para las carreras *STEM*[6]. Esto se debe a que el MINEDUC[7], en su plan de estudio para los colegios desde 1° básico hasta 4° medio, no establece programación como una asignatura obligatoria [MINEDUC, 2018]. Adicionalmente, **la formación de pedagogos especializados en la materia es escasa** en comparación a asignaturas más tradicionales, como lenguaje o matemáticas, por lo que la oferta de profesores en esta área es baja. Como consecuencia, **muchos estudiantes tienen su primer acercamiento a la programación en un curso introductorio de nivel universitario**.

En su trabajo, el académico del departamento de Ingeniería Informática y Ciencias de Computación de la Universidad de Concepción, Jorge López Reguera [López Reguera *et al.*, 2011], menciona que algunos factores que influyen en el éxito de los estudiantes en los cursos de programación son: experiencia previa, conocimiento en otras disciplinas, autoeficacia y motivación.

1.2.1. Experiencia previa

Al contar con experiencia previa en programación, se hace más fácil comprender algunos conceptos básicos, como lo son los tipos de datos, variables, condicionales y ciclos. El hecho de **tener una noción de estos elementos reduce la carga cognitiva** de los estudiantes, al no tener que invertir tanto tiempo en entenderlos, lo que a su vez permite preocuparse más por el razonamiento lógico y la resolución de problemas. Del mismo modo, es una buena forma de **evitar la aparición de errores de concepto**.

Sin embargo, dado el plan de estudios del MINEDUC [MINEDUC, 2018], **son pocos los estudiantes que cuentan con experiencia previa en programación**, siendo un beneficio que se reserva para aquellas personas autodidactas o que provienen de un reducido número de establecimientos de enseñanza media donde sí se enseña programación.

1.2.2. Conocimiento en otras disciplinas

Está comprobado que **tener conocimientos en otras áreas del saber humano**, como las matemáticas [Meza *et al.*, 2024], lenguaje o ciencias naturales, **influye positivamente tanto para el aprendizaje de programación como para el desarrollo del pensamiento computacional**, puesto que ayudan a comprender y analizar los diferentes problemas a los que un estudiante se puede enfrentar, idear soluciones creativas y comunicarlas con otras personas de forma comprensible.

[6] *Science, Technology, Engineering and Math* (Ciencias, Tecnología, Ingeniería y Matemáticas).
[7] Ministerio de Educación.

Sin embargo, también **pueden ser un factor en la aparición de errores de concepto**, ya que hay algunos conceptos que son usados tanto en programación como en otra disciplina pero funcionan de forma diferente, llegando a causar confusión.

1.2.3. Autoeficacia

El psicólogo Albert Bandura define la autoeficacia como "la opinión de una persona sobre sus capacidades de organizar y ejecutar cursos de acción necesarios para lograr los tipos designados de desempeño"[Bandura, 1986]. Dicho de otra forma, corresponde a la auto percepción de una persona sobre su capacidad para resolver un problema determinado.

Una persona con un alto nivel de autoeficacia se ve beneficiada en sus procesos cognitivos y afectivos, lo que hace más probable su persistencia y éxito académico en un curso de programación. Algunas actividades que pueden aumentar el nivel de autoeficacia son el **estudio constante** para adquirir conocimiento y la **práctica**, ya sea mediante ejercicios, tareas o similares, con los cuales generar la sensación de progreso [Dodobara, 2005].

De igual manera, **los profesores tienen un rol fundamental en mejorar la autoeficacia de sus estudiantes**, ya que son los guías durante el proceso de aprendizaje y pueden entregar la orientación necesaria para lograr los objetivos del curso.

1.2.4. Motivación

La motivación es fundamental al momento de aprender a programar, ya que una persona que se encuentra motivada tiene un alto nivel de compromiso, dedicando una mayor cantidad de tiempo a estudiar y practicar; en contraposición a una persona que no está motivada y dedica un mínimo de tiempo a estudiar sobre el tema, lo que a menudo significa un peor desempeño académico.

Al igual que con la autoeficacia, los profesores nuevamente toman un rol de suma importancia para motivar a sus estudiantes; ya sea con mensajes de apoyo directo, enseñando los contenidos de una forma que sea llamativa para los estudiantes, etc.

Si bien este es un factor crucial para la permanencia y buen desempeño en un curso de programación, la realidad nacional es que **los jóvenes no están motivados en aprender programación**, a pesar de ser nativos digitales [Marc, 2001] y disponer de grandes avances tecnológicos. Esta situación se puede evidenciar en el trabajo de Catalina Torrent, quien analizó la iniciativa nacional realizada por el MINEDUC en el año 2017: "Mi Taller Digital de Programación". Este era un taller extracurricular, en el cual los estudiantes aprendían programación por bloques [MINEDUC, 2017]. Sin embargo, Torrent [Torrent, 2019] analizó los datos de participación (tabla 1) y observó que **la mayoría de estudiantes participantes no terminaron el taller**, indicando una falta de motivación para aprender programación.

Tabla 1: Estudiantes que terminaron el taller del MINEDUC en 2017.
Fuente: Catalina Torrent [Torrent, 2019].

Cantidad de estudiantes que terminan el taller	Frecuencia de Establecimientos	Porcentaje
0	211	70,3 %
1-5	53	17,6 %
6-10	15	5 %
11-15	12	4 %
16-20	9	3 %
TOTAL	300	100 %

El MINEDUC está al tanto de este problema y como solución puso en marcha el **Plan Nacional de Lenguajes Digitales** [MINEDUC, s f], con el cual buscan introducir los conceptos de pensamiento computacional y programación a niños y jóvenes, desarrollando las competencias en 4 planes formativos con temas de diferentes áreas de la informática, los que varían según el rango etario. Para lograrlo, este plan abarca la formación de docentes especializados, difusión a través de campañas y realización de torneos públicos, entregando los recursos tecnológicos necesarios.

1.3. Videojuegos en el aprendizaje de la programación

Los videojuegos son un medio de entretenimiento masivo, que cuentan con una amplia gama de géneros y dispositivos en los cuales se pueden jugar. Esta diversidad y accesibilidad hace que sean muy atractivos, sobre todo para los jóvenes. Su uso en una sala de clases se encuentra dentro del marco de la **gamificación**.

El programa de becas del Banco Santander define gamificar (o ludificar) como “llevar el juego a ámbitos donde no es común (...). Consiste en **aprovechar recursos y herramientas propias del juego para mejorar el rendimiento y los resultados académicos**”[Santander, 2023]. Dado el antecedente de que los estudiantes no están motivados por aprender programación, un videojuego se presenta como una solución viable para estimular su interés y así desarrollar las habilidades de pensamiento computacional y programación, mitigando los errores de concepto más comunes en un curso introductorio.

1.4. Objetivos de la Memoria

El desarrollo de esta Memoria contempla los siguientes objetivos:

1.4.1. Objetivo general

Desarrollar un videojuego que apoye el proceso de aprendizaje de programación y ayude a mitigar errores de concepto en un curso introductorio de nivel universitario.

1.4.2. Objetivos específicos

- Determinar los conceptos que podrían apoyarse lúdicamente y diseñar elementos del juego que los desarrollen.
- Definir estrategias dentro del juego que puedan usarse para desarrollar habilidades de programación y mitigar errores conceptuales.
- Evaluar la eficacia del videojuego con estudiantes en el curso de Programación.

1.5. Alcances de la Memoria

El videojuego desarrollado para esta Memoria está pensado para ser utilizado por estudiantes del curso de Programación de la UTFSM, los que en su mayoría no cuentan con experiencia previa en programación. Sin embargo, aquellos estudiantes que con anterioridad hayan tenido un acercamiento, también pueden verse beneficiados.

Se espera que este videojuego sea un aporte al formato de clases tradicional, mas no un reemplazo de las mismas, presentando una cantidad suficiente de actividades que fomenten el desarrollo del pensamiento computacional.

CAPÍTULO 2
MARCO CONCEPTUAL

2.1. Importancia del pensamiento computacional

Se han escrito numerosos artículos sobre el pensamiento computacional, destacando aquellos hechos por la doctora Jeannette Wing [Wing, 2006] [Wing, 2008] [Wing, 2011]. Debido a la gran cantidad de autores que han escrito sobre el tema, **se cuenta con diferentes definiciones de lo que es el pensamiento computacional**, como lo hace notar el académico Xiaodan Tang en su artículo [Tang *et al.*, 2020]. Por este motivo y para efectos de este trabajo, se entenderá como **pensamiento computacional** al **proceso de análisis de un problema y la formulación de sus soluciones, utilizando herramientas de la computación para su resolución**.

Wing enfatiza en que la mayoría de los procesos mentales que se realizan mientras se está resolviendo un problema forman parte de este concepto. Algunas de las características que posee el pensamiento computacional son:

- Va más allá que ser capaz de programar, también requiere pensar en diferentes niveles de abstracción.
- Es una habilidad fundamental que toda persona debería adquirir.
- Es una forma en la que los humanos piensan, siendo inteligentes y creativos.
- Enriquece el pensamiento matemático e ingenieril, pudiendo combinarse con ellos o funcionando como un complemento.

Si bien **el pensamiento computacional abarca una cantidad de habilidades muy amplia**, estas no pueden medirse por separado, sino que **deben ser medidas en grupos de habilidades** [Oliveira *et al.*, 2021]. En este contexto, una de las iniciativas más conocidas en este ámbito es **Bebras** [Bebras, 2004], que busca promover la informática y el pensamiento computacional en actividades conocidas como desafío Bebras, los cuales están diseñados para estudiantes de todas las edades, como se puede apreciar en las figuras 1 y 2. Estos desafíos son utilizados como una herramienta de medición para ciertas habilidades del pensamiento computacional.

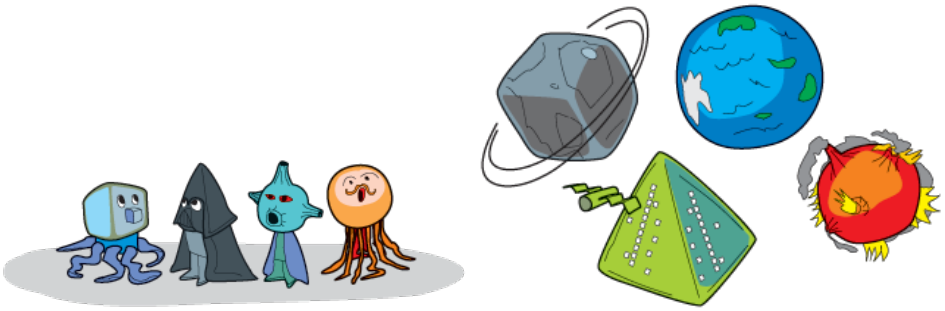


Figura 1: Ejercicio Bebras de ejemplo para niños de 5 a 8 años.
Fuente: Bebras [Bebras, s f].

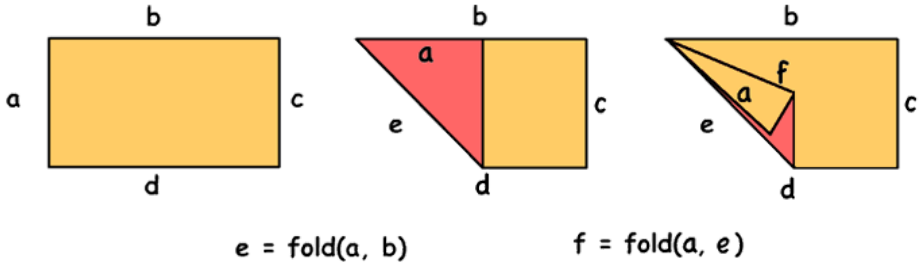


Figura 2: Ejercicio Bebras de ejemplo para jóvenes de 17 a 18 años.
Fuente: Bebras [Bebras, s f].

Debido a la amplitud propia de este concepto, **el pensamiento computacional se ha expandido a otras disciplinas fuera de la informática**, tales como la biología, las humanidades y las matemáticas. Estas áreas del saber humano se han visto beneficiadas de los avances computacionales, permitiendo optimizar sus procesos [Wing, 2011].

De igual modo, **la informática también se ha visto beneficiada de otras disciplinas**, como ocurre en el ámbito educativo. Se ha mostrado que aquellos estudiantes con mejor desempeño en matemáticas también tienen un mejor rendimiento al realizar tests de pensamiento computacional [Mooney y Lockwood, 2020]. Cabe destacar el caso nacional, en el que aquellos estudiantes que obtuvieron mejores resultados en la prueba PAES[8] de matemáticas, tuvieron un buen desempeño en el test de diagnóstico de pensamiento computacional del curso de programación de la UTFSM [Meza *et al.*, 2024].

2.2. Errores de concepto

Los errores de concepto o *misconceptions* son **ideas equívocas que las personas tienen acerca de un tema** [Qian y Lehman, 2017]. Estos errores abarcan desde la concepción de ideas hasta su codificación en algún lenguaje de programación. Su aparición se debe a varios factores, tanto previos como durante el aprendizaje en un curso de programación.

[8]Prueba de Acceso a la Educación Superior.

En el trabajo de Jeffrey Bonar [Bonar y Soloway, 1985] se menciona que uno de los factores que influyen en la aparición de errores de concepto es el **lenguaje natural** de los estudiantes, el que entra en conflicto con las instrucciones formales de un lenguaje de programación.

Otro factor a tener a cuenta es que **los estudiantes confunden el funcionamiento de conceptos que tienen el mismo nombre** (como el de variable, en matemáticas) o enfrentan dificultades porque están aprendiendo **temas que no habían visto con anterioridad**, como aprender un lenguaje de programación. Francisco Vásquez [Vásquez Pinto, 2020] elaboró un listado con los errores de concepto que han sido identificados en curso de programación de la UTFSM, destacando aquellos que pertenecen a las siguientes categorías:

- Conceptuales.
- Asignación de variables.
- Secuencias de control, como condicionales y ciclos.
- Uso de funciones.

Es importante recalcar que **la aparición de errores de concepto es perjudicial para el desarrollo del pensamiento computacional**, ya que al tener ideas erradas sobre ciertos aspectos, los estudiantes terminan creando soluciones equivocadas a los problemas a los que se enfrentan, lo que termina perjudicando la autoeficacia, motivación y las notas de los estudiantes. Por este motivo, **es crucial que los profesores logren detectar y entender los errores de conceptos de sus estudiantes**, para poder corregirlos y encaminarlos en su proceso de aprendizaje.

Lo más común para detectar errores de concepto es realizar evaluaciones (formativas o evaluativas) con las cuales los profesores puedan determinar si los estudiantes aprendieron correctamente los contenidos del curso y precisar aquellos temas con los cuales los estudiantes tienen mayores problemas [Kaczmarczyk *et al.*, 2010]. Algunas evaluaciones cuentan con preguntas que ofrecen **cierto nivel de apoyo** en algunos aspectos para **reducir el nivel de carga cognitiva**. A este tipo de ayuda se le denomina **andamiaje** [Pérez-Pueyo *et al.*, 2019].

2.3. Videojuegos como medio didáctico

Un videojuego es un tipo de software en el cual el usuario (conocido también como jugador) es capaz de interactuar con un dispositivo electrónico que muestra imágenes. Los jugadores pueden realizar acciones que siguen ciertas reglas predefinidas, conocidas como mecánicas, con las cuales pueden causar impacto en el mundo del videojuego. Existen diversos géneros en función del tipo de mecánicas que presentan, tales como juegos de plataformas, juegos de disparos (*shooters*), estrategia, etc. Estas características hacen que hoy en día, los videojuegos sean muy populares, sobre todo en la población joven [García, 2006].

Dada la poca motivación de parte de los jóvenes por aprender programación, implementar un videojuego educativo para reforzar los contenidos vistos en cursos introductorios de programación puede entusiasmarlos a aprender. Cabe destacar que esto **es un complemento al formato de clases tradicional** y no un reemplazo de éstas. Corresponde a una herramienta de gamificación, una técnica educativa que adapta las mecánicas de los videojuegos al ámbito educativo y aprovecha elementos propios de éstos para mejorar el rendimiento y los resultados académicos [Santander, 2023].

2.3.1. Aportes de los videojuegos

Uno de los principales aportes de los videojuegos es brindar un **ambiente controlado**, es decir, el diseño de cada nivel está hecho para ser superado utilizando las mecánicas del juego. Si el jugador se equivoca, puede volver a intentar el nivel las veces necesarias hasta lograrlo [García, 2006]. Esta es una buena manera de **mitigar los errores de concepto** de los estudiantes, porque si no superan los niveles a los que se enfrentan, quiere decir que hay un error en el procedimiento que hicieron, por lo que deberán revisar lo que hicieron para dar con la respuesta correcta.

Otro factor del que se puede sacar provecho de los videojuegos es la **sensación de progreso**, ocasionada cuando el jugador va superando los niveles cuya dificultad va incrementando, lo que evidencia una mejora en su capacidad de resolver problemas y por lo tanto, **mayor desarrollo del pensamiento computacional**. Adicionalmente, a medida que el jugador avanza en los niveles del videojuego, **va ganando motivación y confianza** al resolver los problemas a los que se enfrenta, lo que es beneficioso para su autoeficacia [Czikszentmihalyi, 1990].

2.3.2. Desafíos de los videojuegos

Así como los videojuegos entregan elementos con los cuales los estudiantes se ven beneficiados, también cuentan con grandes desafíos que deben ser capaces de superar para lograr ser un aporte como herramienta educativa.

Un desafío importante para que los jugadores se mantengan jugando es la **dificultad** que tiene. Un videojuego con una dificultad muy elevada hace que la mayoría de jugadores no puedan progresar, llegando a ser tedioso y desmotivante. Por el contrario, si un videojuego es muy fácil de superar, no presenta un desafío para el jugador y puede terminar aburriéndose [Gari *et al.*, 2018].

Del mismo modo, los videojuegos deben tener en claro cual es el **objetivo** que buscan lograr, pues es la meta principal que mantiene el interés del jugador a lo largo de las actividades y debe estar ligado a los objetivos educativos que se buscan lograr [García, 2006].

2.3.3. Ejemplos de videojuegos educativos

A continuación, se muestran algunos ejemplos de juegos educativos utilizados en el ámbito de la informática:

CodeCombat [CodeCombat, 2013] es un videojuego con elementos de *RPG*[9], cómo selección y personalización de personajes, en el cual **se debe programar para que el personaje elegido pueda alcanzar la meta** y completar el nivel. Los jugadores pueden escoger varios mapas para jugar, los que corresponden a distintas categorías, tales como sintaxis de código, construcción de niveles, IA simple, etc. Cabe destacar que **el jugador puede escoger un lenguaje de programación en particular** para los niveles, como por ejemplo: *Python*, *C++* y *Java*.



Figura 3: Imagen de uno de los mapas de *Code Combat*.
Fuente: Videojuego *Code Combat* [CodeCombat, 2013].

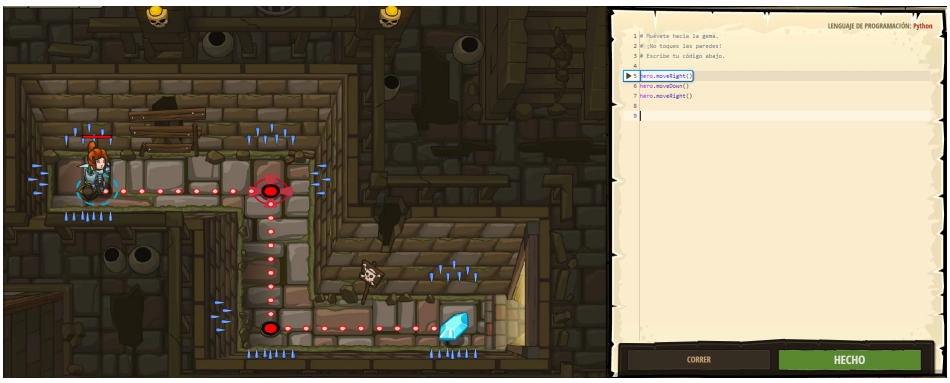


Figura 4: Imagen de un nivel de *Code Combat*.
Fuente: Videojuego *Code Combat* [CodeCombat, 2013].

[9] *Role Playing Game*.

CodeCombat es desarrollado por la empresa homónima, lo que se evidencia en la **buena calidad visual y la extensión del videojuego**, siendo atractivo para los jugadores. Además, los profesores pueden crear sus propias salas de clases virtuales y enfocarlas a un tema en específico. Sin embargo, tiene implementado un **sistema de suscripción de cuentas, lo que crea una barrera de pago** para acceder a más personajes, niveles y características *premium*.

También se cuenta con la campaña de educación conocida como “La Hora del Código”[Kodea, sf], la cual se enfoca en enseñar a niños y jóvenes algunos conceptos de programación mediante la **programación por bloques**. En su sitio web, hay muchos **videojuegos gratuitos** que utilizan elementos llamativos y conocidos (como música popular o el uso de personajes de distintas películas) que sirven como retroalimentación directa de lo que se está haciendo con el código, ya que al ejecutarlo, **se puede ver en una sección de la pantalla como se van realizando las acciones programadas**. Se puede acceder a todo el catálogo de videojuegos a través de su sitio web, lo que facilita su accesibilidad; aunque si se desea, se pueden descargar los ejecutables para jugar sin necesidad de una conexión a Internet. También, cabe destacar que antes de iniciar un nivel, **se muestra información acerca de lo que se pide realizar** y una pequeña ayuda sobre cómo lograrlo, lo que permite mantener el foco en la programación y lo útil que puede ser para resolver problemas cotidianos.

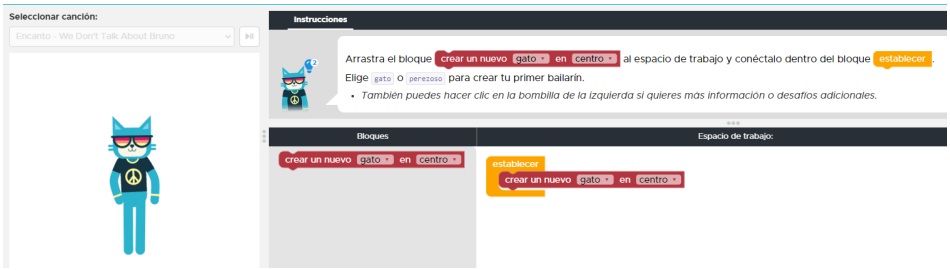


Figura 5: Una actividad de los videojuego de la Hora del Código.
Fuente: La Hora del Código [Kodea, sf].



Figura 6: Otra actividad de los videojuego de la Hora del Código.
Fuente: La Hora del Código [Kodea, sf].

Por otro lado, en el trabajo de José Antonio Lara [Lara Mira, 2022], se muestra la efectividad de un videojuego conocido como GIPPPY para ayudar a los estudiantes de la Universitat Politècnica de València. Se utilizan aspectos básicos de **programación en *Python*** para completar los niveles. Este juego fue altamente valorado por la utilización de **niveles que van incrementando su dificultad** y por la sensación de progreso que causaba en los estudiantes que lo jugaron. Cabe destacar que este videojuego fue utilizado en un pequeño grupo de estudiantes que ya tenía experiencia con este lenguaje de programación y **el autor** [Lara Mira, 2022] **recomienda contar con conocimientos previos** (aunque no necesariamente en profundidad) sobre programación en *Python* para aprovechar al máximo la experiencia con el videojuego.



Figura 7: Nivel 1 del videojuego GIPPPY.
Fuente: [Lara Mira, 2022].



Figura 8: Nivel 9 del videojuego GIPPPY.
Fuente: [Lara Mira, 2022].

Se puede notar que existen varios videojuegos que se utilizan en el ámbito educativo. En cada caso, han demostrado tener éxito ayudando a quienes los juegan, creando un ambiente seguro para que los estudiantes puedan practicar sin temor a equivocarse y motivándolos a adentrarse en la programación. Sin embargo, estos juegos están diseñados principalmente para introducir a los estudiantes en este campo, enfocándose en los elementos básicos y dejando de lado problemas más complejos y los errores conceptuales que los jugadores puedan tener.

Por lo tanto, para aplicar un juego en la UTFSM, es necesario contar con una mecánica llamativa que motive a los estudiantes y que se diferencie de los ejemplos anteriores. Para que el juego sea de utilidad, debe estar alineado con los contenidos impartidos durante el curso de programación, abordando los errores de concepto más comunes cometidos por los estudiantes.

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

3.1. Presentando *Pyconception*

Para mitigar los errores de concepto en estudiantes de la asignatura de programación de la UTFSM, se ha creado el videojuego ***Pyconception***, el cual permite a los estudiantes **reforzar los contenidos vistos en clases** (mas no reemplazarlas) y ayudarlos a desarrollar el pensamiento computacional jugando. El título de este juego nació de la idea de mezclar las palabras *Python*, el lenguaje de programación que se utiliza a lo largo del curso; y *misconception*, pues se pretende abordar los errores de concepto cometidos por los estudiantes utilizando este juego.

Pyconception es un videojuego en 2D donde el jugador controla a un *ninja* que se aventura en múltiples laberintos. En cada nivel **se encuentran dispersas varias llaves de diferentes colores** (rojo, azul, amarillo y verde) **y una puerta de meta que requiere una cantidad determinada de llaves de cada tipo para poder pasar al siguiente nivel**. Cuando el jugador pasa por la puerta de meta, se verifica si posee todas las llaves necesarias y muestra un mensaje de victoria o derrota, según corresponda.



Figura 9: El *ninja* que controla el jugador, los tipos de llaves y la puerta de meta.
Fuente: *Pyconception*.

En este aspecto es donde se ponen a prueba las habilidades de pensamiento computacional del jugador, ya que cuenta con un **panel de código en *Python***, al que se denomina **código principal**, el cual **debe rutear**[10] **para saber cuántas llaves de cada tipo necesita**, aplicando los contenidos vistos en las diferentes unidades del curso. Los nombres de las variables utilizadas hacen referencia a los tipos de llaves necesarios para superar el nivel (por ejemplo, la variable *llavesRojas* indica la cantidad necesaria de llaves de color rojo). Las cantidades necesarias siempre corresponden a los **valores exactos** que tienen todas las variables utilizadas en el código al finalizar el ruteo y los valores finales siempre corresponden a enteros positivos.

[10]Dar seguimiento a una secuencia de líneas de código.

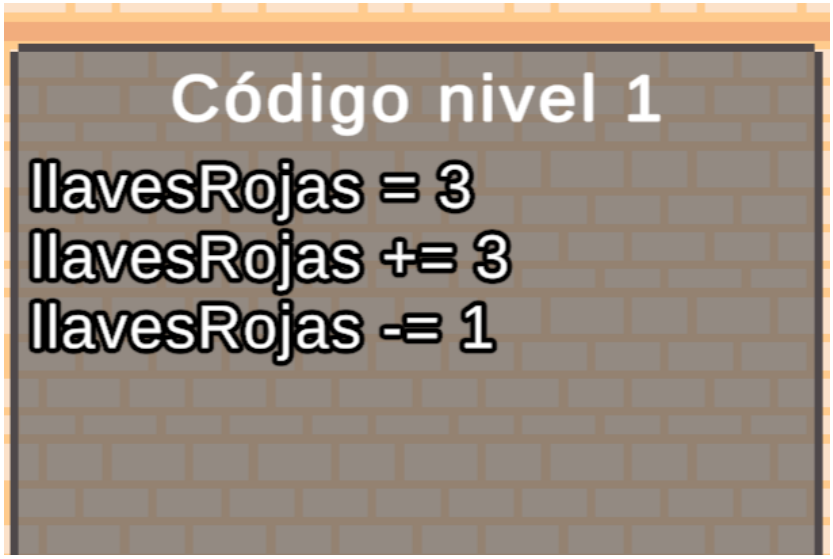


Figura 10: Ejemplo de código principal.
Fuente: *Pyconception*.

El código principal de cada nivel cambia varias veces los valores de las variables que usa, aplicando diferentes operadores que pueden aumentar la dificultad para seguir el ruteo mentalmente. Por esta razón, ***Pyconception* cuenta con un panel de apuntes** (figura 11), permitiendo anotar los valores de las variables mientras se lee el código. Cabe destacar que el uso de esta herramienta es recomendado (sobre todo para niveles avanzados), mas no obligatorio.



Figura 11: Panel de apuntes y ejemplos de uso.
Fuente: *Pyconception*.

El jugador puede moverse libremente utilizando las teclas WASD o las flechas direccionales (figura 12) buscando las llaves necesarias, las cuales puede recoger automáticamente al pasar por encima de ellas. También **cuenta con un panel de inventario** (figura 13) con el que puede hacer seguimiento de las llaves que ha recogido y eliminarlas si lo estima conveniente.

No existe restricción en cuanto al orden en el que se recogen las llaves, siempre y cuando cumpla con la cantidad necesaria. Adicionalmente, el jugador puede usar la rueda del *mouse* (figura 14) para ajustar el nivel de *zoom* de la cámara y tener visión sobre un área más amplia, facilitando la búsqueda de llaves y evitando perderse o dar vueltas en círculos.

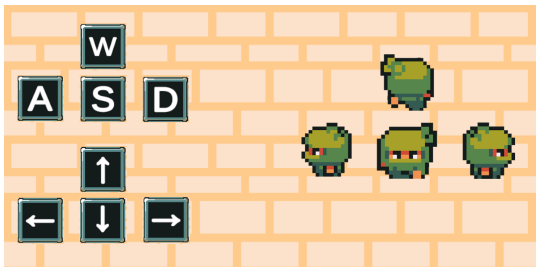


Figura 12: Controles de movimiento del *ninja*.
Fuente: *Pyconception*.



Figura 13: Ejemplos de uso del panel de inventario.
Fuente: *Pyconception*.

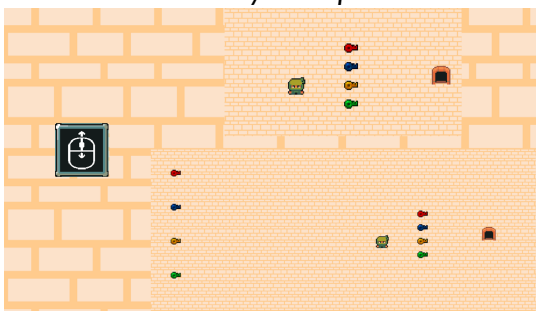


Figura 14: Controles para el *zoom* de la cámara.
Fuente: *Pyconception*.

3.2. Minijuegos

Los niveles 1 y 2 sirven como introducción a *Pyconception*, permitiendo que el jugador se acostumbre a los controles y al funcionamiento del juego. Es a partir del nivel 3 que se incorporan **minijuegos** a los niveles, **los que corresponden a actividades que cuentan con un enunciado y 2 fragmentos pequeños de código que presentan ligeras diferencias entre sí**, por lo que el jugador debe comparar los códigos que se le presentan y determinar cuál de ellos cumple satisfactoriamente el enunciado de la actividad. **Los minijuegos se encuentran en letreros** (figura 15) dispersos a lo largo de cada nivel. Si el jugador se acerca lo suficiente a ellos, aparecerá un icono indicando que puede interactuar con los letreros presionando la tecla *E* para iniciarlo.

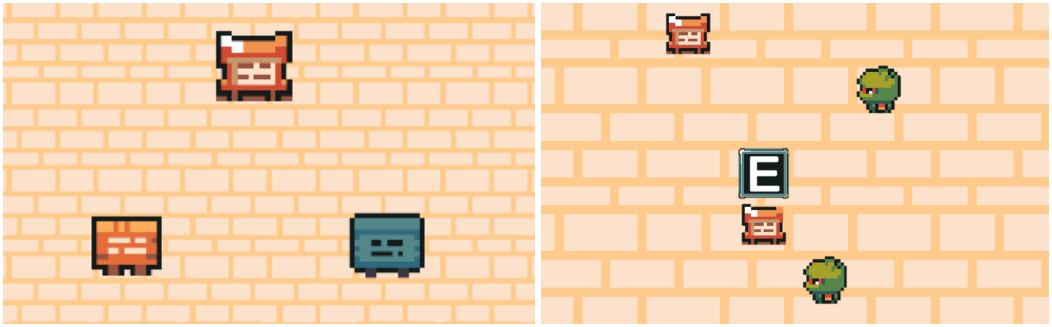


Figura 15: Todos los tipos de letreros y activación de minijuegos.
Fuente: *Pyconception*.

Una vez que el jugador comienza un minijuego (figura 16), se abre un panel que contiene el enunciado, los códigos a comparar, los que cuentan con un botón asociado para responder y un botón *X* que permite cerrar el minijuego sin tener que jugarlo.

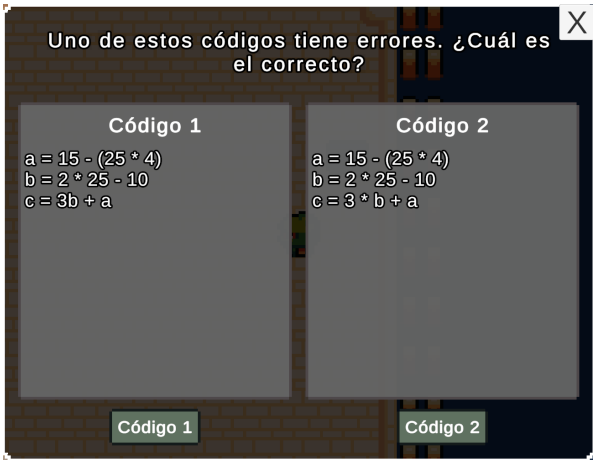


Figura 16: Primer ejemplo de un minijuego.
Fuente: *Pyconception*.

Si el jugador selecciona el código que satisface el enunciado, es recompensado con una llave que aparece cerca del letrero activado. En caso caso contrario, se despliega otro ejercicio que no haya sido respondido correctamente. En ambos casos, se muestra una animación acorde al resultado (figura 17) del minijuego para indicar al jugador si acertó o falló. Cabe destacar que **las llaves otorgadas por los minijuegos son necesarias para superar los niveles**, por lo que el jugador se verá forzado a completarlos para seguir avanzando.

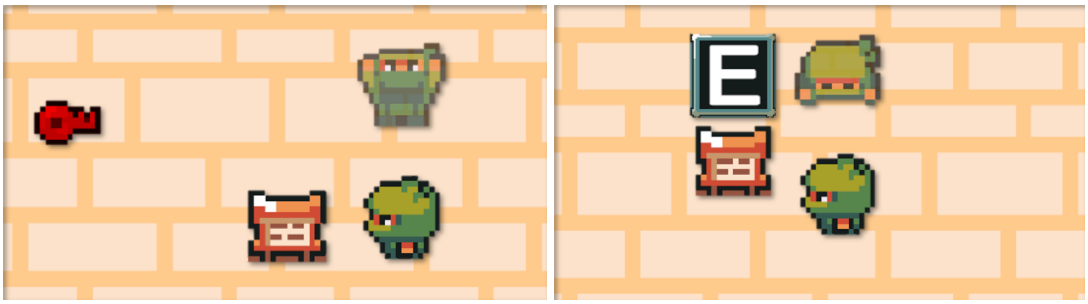


Figura 17: Animación de victoria y derrota al terminar un minijuego.
Fuente: *Pyconception*.

Los minijuegos están enfocados en presentar *misconceptions* de forma intencionada en uno de los códigos para que un estudiante logre identificarlos y superarlos, logrando así un aprendizaje correcto de los conceptos vistos en clases. Cabe destacar que los errores de concepto presentes en los minijuegos provienen o están inspirados por la tabla elaborada por Francisco Vásquez en su trabajo [Vásquez Pinto, 2020].

Por ejemplo, en la figura 16 se puede observar que en la línea 3 del código 1 se tiene la instrucción $c = 3b + a$, que ocasiona un error debido a que falta un operador entre 3 y b, a diferencia de la línea 3 del código 2. **El *misconception* en este código sería pensar que el código 1 es correcto, asumiendo que si se omite un operador entre dos términos se está realizando una multiplicación**, tal como ocurre en matemáticas.

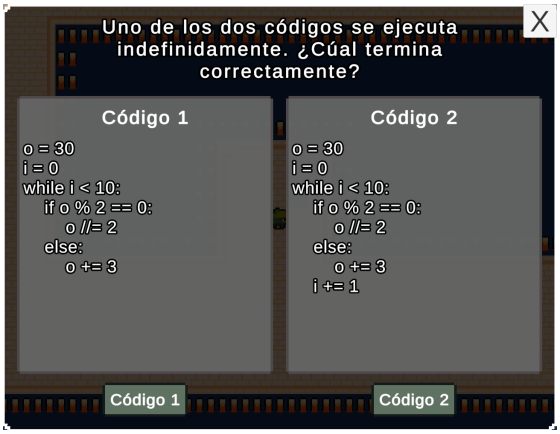


Figura 18: Segundo ejemplo de un minijuego.
Fuente: *Pyconception*.

Otro ejemplo de error de concepto se puede ver en la figura 18, donde cada código ejecuta un ciclo *while* que opera con la variable o mientras i sea menor a 10. Sin embargo, **en el código 1 no se está incrementando la variable i al final de cada iteración del ciclo**, provocando que el programa se ejecute indefinidamente. Esto ocurre generalmente porque **los estudiantes piensan que el ciclo terminará por sí solo**.

También está el ejemplo del minijuego de la figura 19, en la que **se confunden las instrucciones *if* y *while***, pensando que en el código 1 se ejecutará un ciclo, cuando en realidad las operaciones indentadas se realizan una vez.

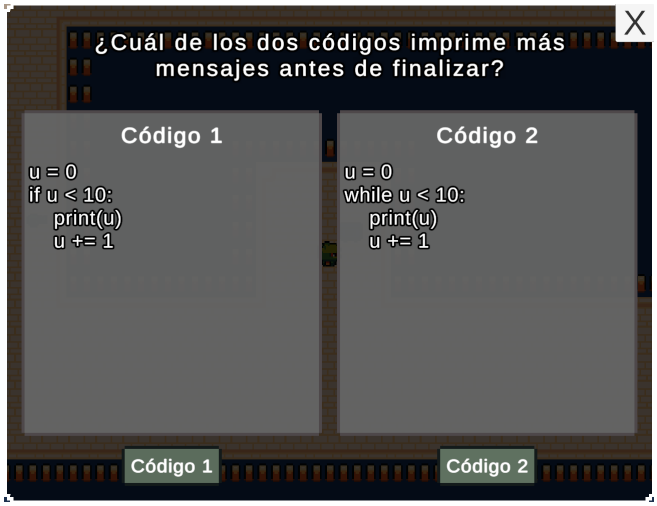


Figura 19: Tercer ejemplo de un minijuego.
Fuente: *Pyconception*.

Existen varios tipos de preguntas en los minijuegos, las que se agrupan en las siguientes categorías:

- Determinar cuál código termina con el menor o mayor valor para una variable al final de la ejecución.
- Uno de los dos códigos contiene un error, por lo que se pide señalar el código que correcto.
- Determinar en cuál código se cumple una condición y por lo tanto, se ejecuta un bloque condicional (*if*) o un ciclo (*while*).
- Uno de los códigos ejecuta un ciclo indefinidamente y se pide identificar aquel que termine la ejecución correctamente.
- Determinar cuál código termina con el mayor largo para una variable de tipo *string*.
- Identificar el código que ejecute una instrucción *print*.
- Determinar cuál código hace un buen uso de funciones.

3.3. Alcance de *Pyconception*

Los contenidos utilizados en *Pyconception* son aquellos que se evalúan en el primer certamen del curso. Cada temática es abordada en una UVA[11] diferente, por lo que el juego destina varios niveles para que cada estudiante pueda reforzar lo aprendido en cada una de ellas. **A medida que se van superando los niveles, la dificultad del juego va en aumento**, tanto por la complejidad incremental de los códigos principales que requieren una mayor atención de los jugadores, como por incluir la materia más reciente vista por los estudiantes. Además, **en los minijuegos se abordan varios errores de concepto colocados intencionalmente**, ya que así se logra mantener la atención del jugador a medida que va progresando y se refuerza lo aprendido en clases.

La distribución de niveles es la siguiente:

- UVA 2: Secuenciales. Abordada en los niveles 1 al 5.
- UVA 3: Condicionales. Abordada en los niveles 6 al 8.
- UVA 4: Ciclos. Abordada en los niveles 9 al 11.
- UVA 5: Strings. Abordada en los niveles 12 al 14.
- UVA 6: Funciones. Abordada en los niveles 15 al 17.

A partir de los niveles correspondientes a la UVA 4, el panel de apuntes también cuenta con espacios para anotar los valores de variables contadoras, tales como i o j. Lo mismo ocurre con el uso del ciclo *for* en la UVA 5. Estos espacios adicionales sólo están presentes si las variables son utilizadas en el código principal de algún nivel.

3.4. Decisiones de diseño

3.4.1. Motor del juego

En videojuegos, un motor corresponde a un conjunto de herramientas que facilitan el desarrollo de un videojuego [Campus, 2023], tales como el renderizado en 2D o 3D, control de animaciones, iluminación, etc. Los motores más populares son *Unreal Engine* [Unreal Engine, 2024], *Godot* [Godot Engine, 2024] y *Unity* [Unity Technologies, 2024].

Para el desarrollo de *Pyconception* se utilizó el motor *Unity* en su apartado 2D, ya que es fácil de usar, cuenta con una documentación robusta para todas las funcionalidades que ofrece y además permite exportar los juegos a múltiples plataformas.

[11] Unidad Virtual de Aprendizaje

3.4.2. Sonidos en el juego

Una sala de clases puede ser un ambiente ruidoso debido a varios factores que pueden afectar negativamente la concentración de los estudiantes, como los son el ruido generado por los estudiantes dentro del aula como fuera de ella, el ruido ambiente, etc. Es por esta razón que ***Pyconception* no cuenta con ningún sonido**, dejando que toda la retroalimentación sea entregada al jugador visualmente, tal como hizo Sebastián Muñoz [Muñoz, 2023] en su Memoria.

3.4.3. Interacción del jugador con el juego

Desde el inicio del desarrollo de *Pyconception* surgió la idea de permitir que el jugador pudiera elegir qué hacer en el juego, para evitar que el flujo del mismo sea lineal y monótono. Para lograrlo, se creó el **menú principal** (figura 20) del juego, que contiene el título del juego y 4 opciones seleccionables por medio del *mouse*:

- **Continuar**: Reanuda la partida desde el último nivel desbloqueado por el jugador.
- **Nueva partida**: Inicia el juego desde el primer nivel.
- **Seleccionar nivel**: Abre el menú de selección de niveles (figura 21), permitiendo al jugador elegir sólo los niveles que ha desbloqueado. Los niveles que se encuentren bloqueados se ven más oscuros para indicar al jugador que no puede acceder a ellos. Cada nivel se encuentra agrupado según la UVA que aborda.
- **Salir del juego**: Cierra el juego.



Figura 20: Menú principal.
Fuente: *Pyconception*.



Figura 21: Menú de selección de niveles.
Fuente: *Pyconception*.

En adición al menú principal, cada nivel posee en la esquina inferior izquierda un panel con 4 botones (figura 22), cuyas funcionalidades son las siguientes:

- Botón para alternar la visibilidad del panel de apuntes.
- Botón para alternar la visibilidad del panel de código.
- Botón para alternar la visibilidad del panel de inventario.
- Botón de pausa. Al hacerle *click*, oculta todos los paneles del juego y despliega un menú de pausa (figura 23) con las siguientes opciones:
  - Reanudar: Quita el menú de pausa y vuelve a mostrar todos los paneles anteriormente activos.
  - Reiniciar nivel: Reinicia el nivel actual.
  - Menú principal: Redirige al jugador al menú principal.



Figura 22: Panel de botones.
Fuente: *Pyconception*.

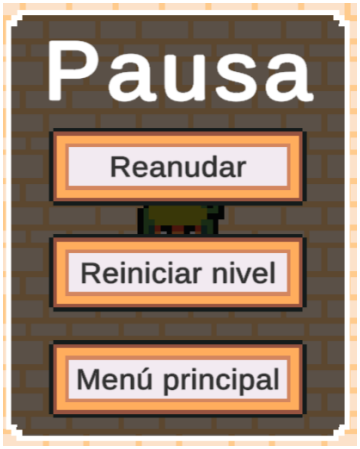


Figura 23: Menú de pausa.
Fuente: *Pyconception*.

3.4.4. *Logs*

En cada nivel del juego **se generan *logs* de forma automática**, los que llevan registro de las llaves necesarias para superar el nivel y de las llaves que el jugador tiene en su inventario cuando alcanza la meta o realiza alguna acción que afecte al desarrollo del nivel, como reiniciar el nivel o volver al menú principal. Estos *logs* **son archivos de texto** (figura 24) que se guardan en una carpeta homónima dentro de los archivos del juego, cuyo nombre sigue el formato *Logs nivel XX* (donde *XX* corresponde al número del nivel jugado), para permitir identificarlos fácilmente. Estas son herramientas de utilidad para los docentes, ya que les permite **hacer seguimiento** de las llaves recogidas por los jugadores, para así **determinar los errores que cometieron** al rutear el código principal y posteriormente corregir aquellos aspectos en los que fallaron.

```
Nivel 1


Cantidades necesarias para pasar el nivel:
Llave Roja: 5


Intento 1 (19:25:21):

Jugador selecciona ir al menu principal.

Intento 2 (19:33:57):
Llave Roja: 5
```

Figura 24: Ejemplo de un *log* del nivel 1.
Fuente: *Pyconception*.

3.4.5. Primera versión de *UI*

Durante las primeras etapas del desarrollo de *Pyconception*, **se había contemplado el uso de una barra de progreso** que indique el porcentaje de llaves necesarias que ha conseguido el jugador, junto con un mensaje para indicar cuáles llaves eran el objetivo del nivel. Estos elementos se encontraban ubicados en la esquina superior izquierda de la *UI* [12] (figura 25) y la barra se actualizaba cada vez que el jugador recogía o eliminaba una llave. Sin embargo, **este apartado fue posteriormente eliminado del juego** por las siguientes razones:

- Se llegó a la conclusión de que su uso podía evitar que los estudiantes tuvieran que rutear el código principal, ya que podían recoger llaves hasta eventualmente llegar al 100 % y superar los niveles; impidiendo que *Pyconception*cumpla con su objetivo de ayudar a mitigar errores de concepto.
- En niveles avanzados, el mensaje con el objetivo se volvería redundante, ya que desde el tutorial se indica claramente que el jugador debe recoger todas las llaves mencionadas en el código principal. Por lo tanto, volver a señalar el objetivo sería más una molestia que un aporte.



Figura 25: Primera versión de *UI* para *Pyconception*.
Fuente: *Pyconception*.

Luego de varias pruebas con los niveles 1 a 5, se determinó que **a los jugadores se les podría complicar llevar el registro de los valores de las variables** al rutear el código principal, puesto que deben realizar más operaciones de mayor complejidad a medida que progresan. Como solución a este problema, se solía utilizar una hoja de cuaderno para realizar el ruteo. Sin embargo, todavía era necesaria una forma de facilitar este trabajo, llegando finalmente a la creación **panel de apuntes**, para así permitir a los jugadores tener un símil de un cuaderno dentro del mismo juego.

[12] *User Interface (Interfaz de usuario)*

Se decidió ubicar a esta herramienta en la zona superior izquierda de la *UI*. Posteriormente, con la implementación del panel de botones, se agregó un botón en la zona inferior derecha para darle al jugador la opción de mostrarla u ocultarla según lo prefiera. La figura 26 muestra la versión final de la *UI* de *Pyconception*.



Figura 26: Última versión de *UI* para *Pyconception*.
Fuente: *Pyconception*.

3.4.6. Paneles tutoriales

Dado que *Pyconception* tiene varios controles, es posible que los jugadores pasen por alto algunos de ellos que pueden ayudarlos a superar los niveles. Para evitar este problema, se incorporaron al juego varios paneles con tutoriales (figura 27) que describen detalladamente cómo utilizar las diversas funcionalidades.

Para que el jugador preste atención al tutorial, se le bloquea el movimiento y se le abre un gran panel en el centro de la pantalla. Este panel cuenta con las siguientes características:

- Título del tutorial.
- Texto en la zona superior del panel, que sirve como introducción al tutorial.
- Texto en la zona central del panel, que sirve para describir el funcionamiento de alguna mecánica. Puede ser reemplazado por una imagen, si se requiere.
- Texto en la zona inferior del panel, utilizado para dar algún consejo o explicación adicional.

Los tutoriales pueden contener varias páginas, las que se pueden cambiar con las flechas laterales de color blanco, ubicadas a los costados del panel. Una vez que el jugador ha visto todas las páginas del tutorial y presione la flecha de la derecha, el tutorial termina y el jugador puede volver a moverse.

El primer tutorial se encuentra en el nivel 1. En él, se explica el objetivo del *Pyconception*, junto con los controles básicos del juego. Luego en el nivel 3, se muestra el tutorial correspondiente a los minijuegos. Finalmente, en el nivel 9, se explica la actualización al panel de apuntes para los niveles que utilizan un contador para ciclos *while*, como *i* y/o *j*.

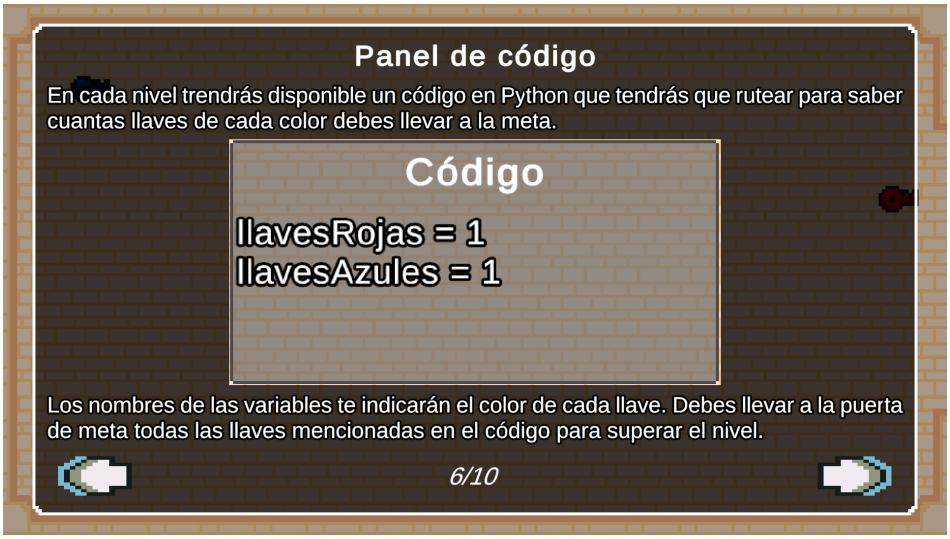


Figura 27: Ejemplo de tutorial.
Fuente: *Pyconception*.

CAPÍTULO 4
VALIDACIÓN DE LA SOLUCIÓN

4.1. Descripción del proceso

Para comprobar si *Pyconception* cumple con su labor de ayudar a mitigar errores de concepto, **se puso a prueba con estudiantes de primer año** del curso IWI-131, correspondiente a Programación. Este proceso se realizó durante el primer semestre del año 2024.

Antes de dar inicio a las clases, durante los meses de enero y febrero, se discutió en conjunto con el profesor guía y la profesora correferente la forma de verificar la eficacia del juego, además del grupo estudiantil con el que se probaría.

Se determinó que la mejor opción era **aprovechar las instancias de ayudantía**, pues los estudiantes ya habrían tenido una sesión de cátedra en la que habrían visto los contenidos de la UVA correspondiente a esa semana en la planificación del curso. También se estableció que **el grupo experimental elegido para realizar las pruebas serían estudiantes de la carrera de Ingeniería Civil Mecánica**, ya que históricamente han tenido un desempeño más bajo que el promedio; lo que se debe a que sus estudiantes no logran comprender los conceptos vistos en clases, resultando en *misconceptions*.

También se concordó en que era necesario contar con un grupo de control (de la misma carrera) que no utilizara el juego y tuviera una ayudantía tradicional. **Para determinar que ambos grupos tuvieran bases similares al momento de ingresar a la universidad**, se optó por utilizar los resultados del **test de pensamiento computacional** que se realiza en las primeras clases del curso, cuyos resultados se encuentran en la tabla 2:

Tabla 2: Resultados del test de pensamiento computacional.
Fuente: Coordinación de Programación, DI UTFSM.

	Grupo experimental	Grupo de control
Promedio	60,83	68,75
Desviación estándar	20,79	21,6
N	20	24

Si bien se aprecia un desempeño ligeramente mejor en el caso del grupo del control, **la diferencia no es significativa**. Por lo tanto, se puede afirmar que ambos grupos iniciaron en la asignatura con un desempeño similar y comparable para medir la eficacia de *Pyconception*.

Además, **a ambos grupos se les tomaría un pequeño test de salida** (Anexo A) **de 3 preguntas con opción múltiple, donde cada pregunta tiene una única alternativa correcta, mientras que las demás presentan errores de concepto**. Estos tests están basados en los inventarios de concepto utilizados por Caceffo [Caceffo *et al.*, 2018] para cursos introductorios de

programación. Posteriormente, se revisarían los test de ambos paralelos y se compararían los resultados. El comportamiento esperado es que el grupo experimental tenga un mejor desempeño en comparación al grupo de control.

A inicios de marzo, antes de la semana de clases de la UVA 2 (secuenciales), se establecieron los grupos con los que se iba poner a prueba este trabajo. **El grupo experimental corresponde al paralelo 201**, siendo los que jueguen *Pyconception*; mientras que **el grupo de control, que corresponde al paralelo 209**, mantiene un modelo de ayudantía tradicional. Cabe destacar que todos **los participantes dieron su consentimiento informado** y en todo momento se les aclaró que la actividad era de carácter voluntario.

En las ayudantías correspondientes al inicio de cada UVA, se destinaron aproximadamente 30 minutos para que el grupo experimental jugara *Pyconception* (a excepción de la primera UVA, para la que se utilizaron alrededor de 50 minutos). Posteriormente, se tomó el post test y el tiempo restante se destinó para responder dudas que los estudiantes pudieran tener. El grupo de control, por su parte, dedicó la totalidad de la ayudantía para exponer el material preparado para la sesión, utilizando los últimos minutos para tomar el mismo post test.

4.1.1. *Hardware* utilizado en ayudantías

Para todas las ayudantías, se utilizó una sala de computación de la UTFSM, donde cada estación cuenta con un computador y los implementos necesarios para su uso (*mouse*, teclado, monitor, etc.). Además, todos los computadores cuentan con las mismas especificaciones, las cuales se detallan a continuación:

- **Sistema operativo**: *Windows* 11 Pro de 64 *bits*.
- **Procesador**: *Intel i5* 12500 a 3.0 [GHz].
- **Tarjeta de video**: 2 tarjetas *NVIDIA T400*, de 4 [GB] de VRAM cada una.
- **Memoria RAM**: 16 [GB].
- **Memoria secundaria**: Unidad de estado sólido *Samsung NVMe PM9A1* de 512 [GB].

4.2. Análisis cuantitativo de resultados

Dado que ambos grupos contaban con estudiantes de diversas carreras, **a todos los estudiantes se les permitió participar de la actividad** correspondiente a cada grupo. Sin embargo, para el análisis de resultados, **sólo se consideraron los test rendidos por estudiantes de primer año pertenecientes a la carrera Ingeniería Civil Mecánica que hayan ingresado a la UTFSM el 2024**.

A las preguntas se les otorgó un puntaje arbitrario para obtener una nota de 0 a 100, donde 0 es la nota más baja y 100 la nota más alta. Para evaluar la significancia estadística de la diferencia entre ambos grupos, se aplicó un t-test.

4.2.1. UVA 2: Secuenciales

La distribución de puntaje para la unidad de secuenciales fue la siguiente:

- 33 puntos para la pregunta 1.
- 34 puntos para la pregunta 2.
- 33 puntos para la pregunta 3.

Los resultados indican que el grupo experimental obtuvo en promedio un 90, con una desviación estándar de 21,95; mientras que el grupo de control obtuvo en promedio un 75, con una desviación estándar de 26,10. Por lo tanto, en la primera sesión donde los estudiantes jugaron *Pyconception*, **el grupo experimental mostró mejores resultados en promedio y un desempeño más consistente**. En este caso, los resultados del t-test indican que **la diferencia entre los resultados de ambos grupos es significativa** ($p = 0.04 \leq 0.05$).

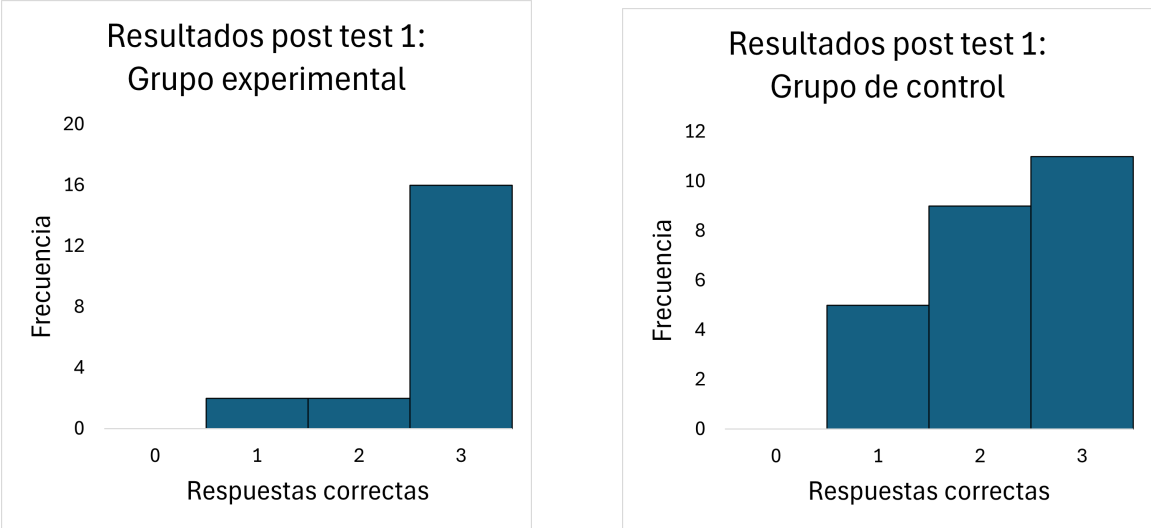


Figura 28: Resultados de ambos grupos en el post test 1.
Fuente: Elaboración propia.

4.2.2. UVA 3: Condicionales

La distribución de puntaje para la unidad de condicionales fue la siguiente:

- 33 puntos para la pregunta 1.
- 33 puntos para la pregunta 2.
- 34 puntos para la pregunta 3.

Los resultados indican que el grupo experimental obtuvo en promedio un 62, con una desviación estándar de 24,25; mientras que el grupo de control obtuvo en promedio un 61, con una desviación estándar de 22,59. En esta unidad, **los estudiantes del grupo experimental mostraron resultados ligeramente mejores que el grupo de control**. Sin embargo, los resultados son considerablemente más bajos si se comparan con los resultados que obtuvieron en la unidad de secuenciales. Esta disminución se atribuye a que a los estudiantes tienen grandes dificultades para comprender las diferentes formas de uso de los condicionales, por lo que requieren un mayor refuerzo en los contenidos de esta UVA.

Los resultados del t-test sugieren que **la diferencia entre los resultados de ambos grupos no es estadísticamente significativa** ($p = 0.86 \geq 0.05$).

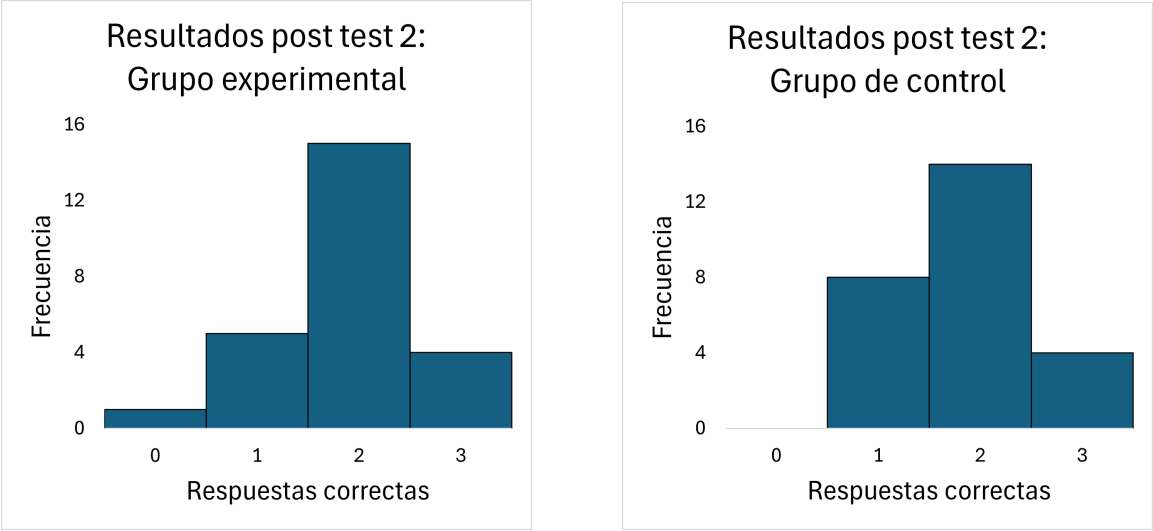


Figura 29: Resultados de ambos grupos en el post test 2.
Fuente: Elaboración propia.

4.2.3. UVA 4: Ciclos

La distribución de puntaje para la unidad de ciclos fue la siguiente:

- 33 puntos para la pregunta 1.
- 33 puntos para la pregunta 2.
- 34 puntos para la pregunta 3.

Los resultados indican que el grupo experimental obtuvo en promedio un 80, con una desviación estándar de 23,26; mientras que el grupo de control obtuvo en promedio un 76, con una desviación estándar de 25,74. **Los estudiantes pertenecientes al grupo experimental consiguieron mejores resultados que el grupo de control** y también mejoraron con respecto a sus resultados de la UVA de condicionales.

Los resultados del t-test sugieren que **la diferencia entre los resultados de ambos grupos no es estadísticamente significativa** ($p = 0.60 \geq 0.05$).

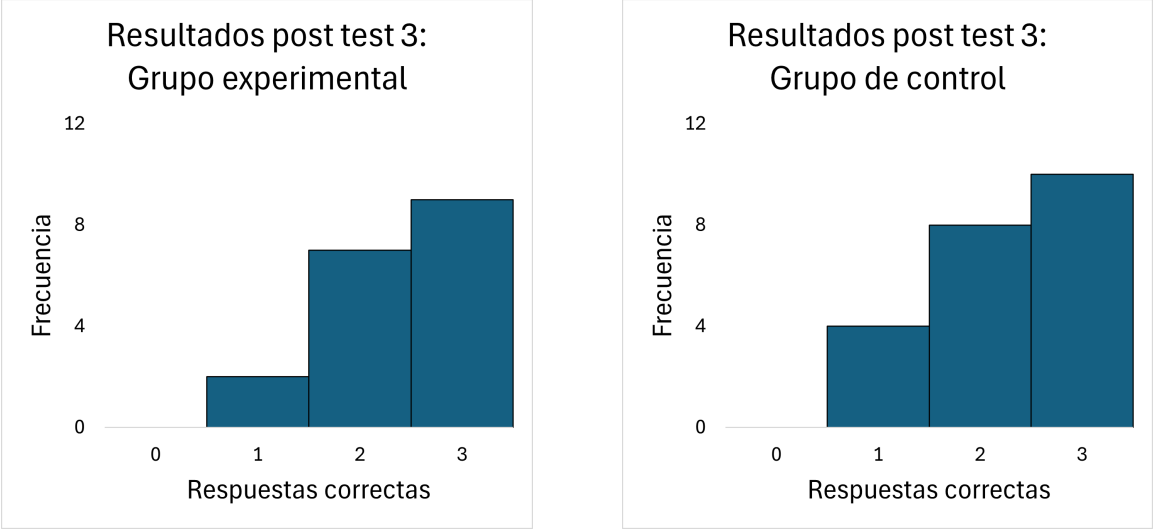


Figura 30: Resultados de ambos grupos en el post test 3.
Fuente: Elaboración propia.

4.2.4. UVA 5: *Strings*

La distribución de puntaje para la unidad de *strings* fue la siguiente:

- 33 puntos para la pregunta 1.
- 33 puntos para la pregunta 2.
- 34 puntos para la pregunta 3.

Los resultados indican que el grupo experimental obtuvo en promedio un 94, con una desviación estándar de 13,75; mientras que el grupo de control obtuvo en promedio un 85, con una desviación estándar de 18,17. Los resultados del t-test sugieren que **la diferencia entre los resultados de ambos grupos no es estadísticamente significativa** ($p = 0.28 \geq 0.05$).

Si bien los resultados muestran que los estudiantes del grupo experimental consiguieron mejores resultados que el grupo de control y consiguieron una mejora significativa con respecto a sus resultados de la UVA de ciclos, **la información obtenida no es confiable, debido al bajo número de participantes en ambos paralelos**.

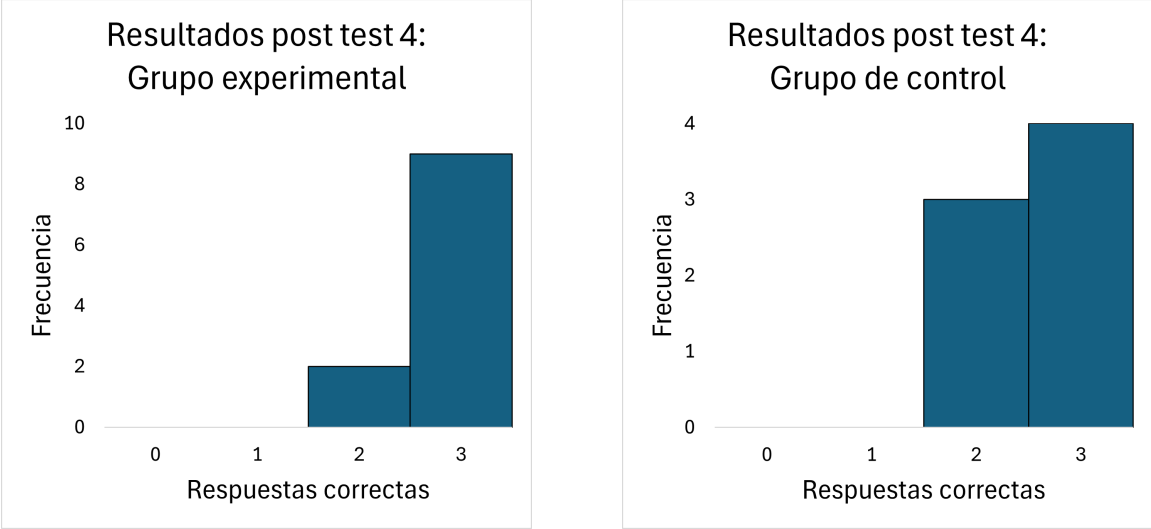


Figura 31: Resultados de ambos grupos en el post test 4.
Fuente: Elaboración propia.

4.2.5. UVA 6: Funciones

La distribución de puntaje para la unidad de funciones fue la siguiente:

- 33 puntos para la pregunta 1.
- 34 puntos para la pregunta 2.
- 33 puntos para la pregunta 3.

Los resultados indican que el grupo experimental obtuvo en promedio un 86, con una desviación estándar de 21,51; mientras que el grupo de control obtuvo en promedio un 65, con una desviación estándar de 31,23. En esta última experiencia, **se observa una marcada diferencia en favor del grupo experimental**, quienes además de subir su rendimiento en comparación a la unidad de ciclos, también **presentan resultados muy homogéneos**.

En este caso, los resultados del t-test indican que **la diferencia entre los resultados de ambos grupos es significativa** ($p = 0.04 \leq 0.05$).

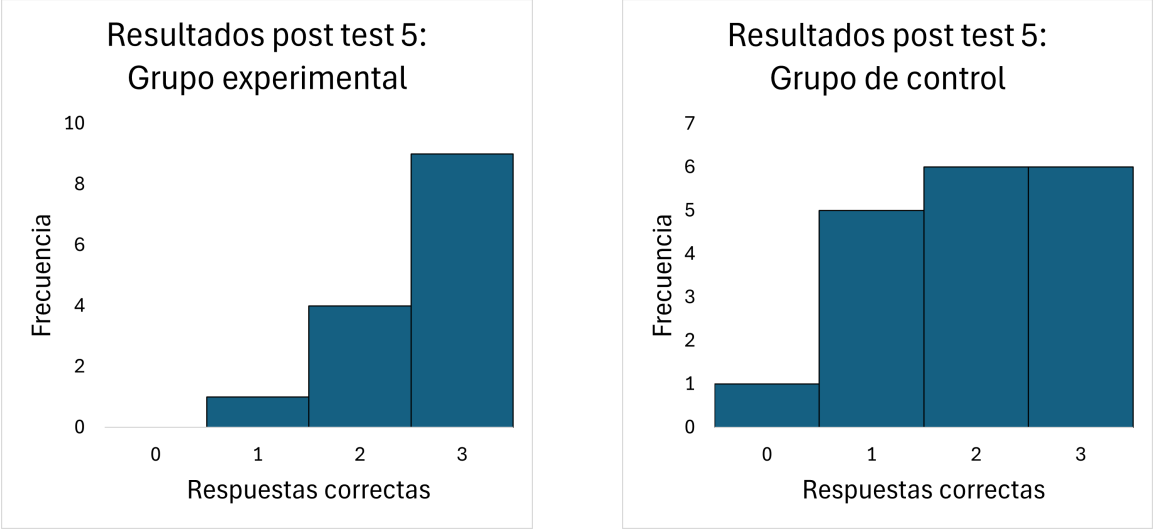


Figura 32: Resultados de ambos grupos en el post test 5.
Fuente: Elaboración propia.

4.2.6. Desempeño de los grupos

Si bien ambos grupo obtuvieron buenos resultados en cada post test realizado, tal como se evidencia en los resultados de cada unidad, **el grupo experimental siempre obtuvo un mejor rendimiento que el grupo de control**. **El punto de más bajo desempeño para ambos grupos fue la UVA 3**, lo que apunta a que es la unidad en la que los estudiantes presentan mayor cantidad de errores de concepto.

A modo de resumen, la figura 33 muestra las líneas de tendencia de cada grupo:

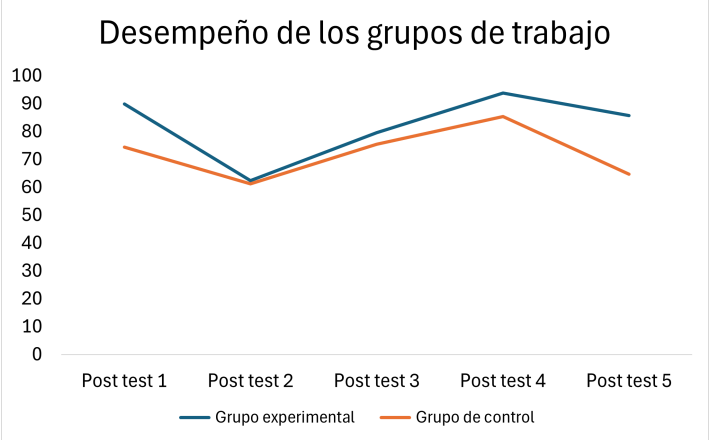


Figura 33: Líneas de tendencia de ambos grupos en los post test.
Fuente: Elaboración propia.

4.3. Test cualitativo

Luego de que los estudiantes del grupo experimental jugaran todos los niveles de *Pyconception*, en ambos grupos se realizó un test cualitativo (Anexo B) con el fin de **obtener información adicional con respecto a la autoeficacia del estudiantado** de ambos paralelos. Además, en el caso del grupo experimental, también se les preguntó sobre el impacto que tuvo *Pyconception* en sus inicios en el curso de programación. Para este test, **se utilizó la escala de Likert**, debido a que permite obtener información clara sobre la opinión de los participantes [Fabila Echauri *et al.*, 2012]. Cada pregunta cuenta con 5 posibles respuestas que van desde "muy en desacuerdo" hasta "muy de acuerdo". Adicionalmente, **al grupo experimental se le hicieron 3 preguntas extra** para que evaluaran su experiencia con *Pyconception*. Las respuestas del grupo experimental se encuentran en la tabla 3 y las del grupo de control están en la tabla 4.

Tabla 3: Respuestas al test cualitativo del grupo experimental.
Fuente: Elaboración propia.

N° pregunta	Muy en desacuerdo	En desacuerdo	Neutral	De acuerdo	Muy de acuerdo
1	1	3	4	5	1
2	0	1	4	6	3
3	0	0	3	10	1
4	0	0	5	4	5
5	0	2	2	5	5
6	0	1	7	6	0
7	0	0	0	7	7
8	0	2	3	6	0
9	0	0	0	8	6
10	0	0	0	6	8
11	0	0	0	1	13
12	0	0	0	4	10

Tabla 4: Respuestas al test cualitativo del grupo de control.
Fuente: Elaboración propia.

N° pregunta	Muy en desacuerdo	En desacuerdo	Neutral	De acuerdo	Muy de acuerdo
1	2	6	9	5	1
2	2	6	9	5	1
3	1	3	9	7	3
4	0	1	7	10	5
5	0	3	5	7	8
6	0	7	3	6	7
7	2	3	9	8	1
8	0	1	8	5	9
9	3	1	11	7	1
10	0	0	2	9	12

A partir de los resultados, se puede observar que **en el grupo de control muchos estudiantes enfrentan dificultades** en varios aspectos. Estos incluyen planificar la resolución de problemas (tanto en las etapas como en las variables involucradas), escribir código en *Python* y corregir su código cuando se les explican sus errores. Además, también presentan problemas relacionados con hábitos de estudio, como la organización y distribución del tiempo.

La mayoría de estos problemas se atribuyen a la **poca experiencia de los estudiantes en programación**, debido a que no han tenido ningún curso afín antes de ingresar a la educación superior. Esta inexperiencia también es un factor que **puede llevar a errores de concepto** al momento de programar, generando dudas e inseguridad sobre el procedimiento planteado

para resolver un ejercicio. **Si estos errores no son detectados y corregidos oportunamente, perjudican la autoeficacia y motivación de los estudiantes**.

Por otro lado, si bien **los estudiantes del grupo experimental** también afirman tener dificultades con sus hábitos de estudio, **poseen una mirada más optimista de sus capacidades**, puesto que la mayoría se percibe capaz de realizar las tareas que se le complican al grupo de control. También **evaluaron muy bien a *Pyconception*** en las preguntas adicionales, valorándolo positivamente como un complemento a su aprendizaje, por su dinamismo y por lo entretenido que les pareció jugarlo; **asegurando que los contenidos vistos en el juego les serán de utilidad durante el curso**.

La razón principal de estos contrastes apunta a ser *Pyconception*, ya que ha significado una experiencia lúdica y didáctica con la que poder practicar los contenidos vistos en las cátedras. Además, **los estudiantes han podido corregir sus errores de concepto por medio del juego**, ya que un error en el ruteo del código principal implica llevar una cantidad incorrecta de llaves a la meta, lo que les impide avanzar de nivel, por lo que deben revisar su procedimiento y corregirlo. Los minijuegos también cumplen un rol fundamental **comparando códigos que presentan *misconceptions* con otros que no los tienen**, dejando en claro a los estudiantes la manera correcta de escribir y ejecutar código en *Python*.

4.4. Observaciones de las experiencias

Durante las sesiones de ayudantía del grupo experimental, se notaron algunos comportamientos por parte de los participantes en los que demostraban su interés por *Pyconception*, los cuales se presentan a continuación:

- Un grupo de estudiantes llegaba a la sala antes de la hora de inicio de la ayudantía para poder jugar por más tiempo.
- Algunos estudiantes llegaban a la sala preguntando si iban a jugar esa semana.
- Los estudiantes comentaban entre ellos cómo superar ciertos niveles.
- Los estudiantes preguntaban si habría otra sesión de *Pyconception* luego de la última (correspondiente a funciones).
- Los estudiantes declararon en otro trabajo que trata sobre mejorar una plataforma usada en el curso que preferían *Pyconception*.

CAPÍTULO 5
CONCLUSIONES

El pensamiento computacional es una habilidad fundamental para desenvolverse en el mundo moderno; sobre todo en el ámbito ingenieril, ya que día a día los ingenieros deben enfrentarse a problemas complejos que requieren pensar en soluciones creativas para resolverlos satisfactoriamente. La tecnología, por su parte, ha ido evolucionando enormemente, adquiriendo mayor protagonismo en el quehacer diario.

Esta habilidad está estrechamente relacionada con la programación debido a los aspectos inherentes de la resolución de problemas. Sin embargo, en Chile, **el pensamiento computacional no se fomenta ampliamente**, ya que el MINEDUC no incluye programación como una asignatura obligatoria en los niveles de enseñanza básica y media [MINEDUC, 2018]. Su introducción se limita a iniciativas específicas, como el Plan Nacional de Lenguajes Digitales [MINEDUC, s f], quedando a cargo de las instituciones de educación superior, donde la mayoría de estudiantes tienen su primer curso de programación. Otro factor relevante es la **baja motivación en aprender programación por parte la población joven** [Torrent, 2019], lo que se traduce en poco interés en desarrollar el pensamiento computacional en carreras no relacionadas a la informática.

Un gran problema que ocurre durante el aprendizaje es la aparición de *misconceptions*, que impactan directamente en la motivación y la autoeficacia de los estudiantes. Para abordar esta problemática, en este trabajo se propuso implementar el videojuego *Pyconception*, buscando mitigar los errores de concepto más comunes en los estudiantes del curso de programación de la UTFSM. A continuación, se detallan las principales conclusiones de este trabajo.

5.1. Gestión y desarrollo del videojuego

Desde un inicio, **se pensó en *Pyconception* como un videojuego de apoyo** para los estudiantes que están empezando un curso de programación y que los motive a adquirir los contenidos del curso. Para lograrlo, era crucial que el juego abordara los mismos contenidos que se veían en una semana determinada.

El proceso de desarrollo fue extenso pero constante. La principal razón fue que el trabajo fue realizado por una persona, lo que comprende la planificación del proyecto, investigación, desarrollo del juego y sus características, elaboración de los post test y la validación de la propuesta. Sin embargo, **se logró cumplir con la calendarización en los plazos establecidos** debido a la continua comunicación con el profesor guía.

La parte de mayor complejidad ocurrió al inicio del proyecto, buscando la forma en la que los jugadores pudieran relacionar los contenidos del curso con una jugabilidad sencilla de entender. Finalmente se llegó a una idea concreta, la mecánica de recolección de llaves, logrando así una base sólida sobre la cual trabajar e iterar en el futuro.

Si bien el apartado gráfico puede ser considerado simple en comparación con producciones de grandes estudios, logró llamar lo suficiente la atención de los estudiantes para que se aventuren a jugar. **Un beneficio directo de la simpleza gráfica de *Pyconception* es que puede ser ejecutado en una gran cantidad de computadores**, incluyendo a aquellos con recursos limitados de *hardware*.

El proceso de desarrollo del videojuego duró aproximadamente 5 meses, destinando los primeros 3 a la creación de las mecánicas principales y el resto para la creación de los niveles y la corrección de *bugs*. El resultado final fue un juego que abarcó todas las unidades que se tenían previstas, una jugabilidad sencilla de entender y una interfaz fácil de usar.

5.2. Determinar *misconceptions*

En el proceso para seleccionar los errores de concepto que se utilizarían en *Pyconception*, **se buscaba que estos cumplieran con 2 condiciones**: **Ser un error presente en la UVA en la que se estaba trabajando en el juego** y **ser un error históricamente común entre los estudiantes del curso**. Para este proceso se conversó activamente con el profesor guía para contar con una buena cantidad de *misconceptions* por unidad.

5.2.1. Incorporación de *misconceptions* al juego

Para que *Pyconception* cumpliera con su función de ayudar a mitigar errores de concepto, **era necesario crear situaciones en las que dichos problemas estuvieran presentes**. Dado que los estudiantes debían rutear el código principal de cada nivel, no era factible dejar los errores ahí, ya que podía ser contraproducente. Por lo tanto, **se optó por usar los minijuegos para mostrar los *misconceptions*. Esta estrategia resultó ser efectiva**, ya que los estudiantes necesitan las llaves que los minijuegos otorgan para pasar al siguiente nivel, lo que hace obligatorio jugarlos.

Además, estas actividades cuentan con un **alto nivel de andamiaje**, puesto que el enunciado deja en claro lo que se pide para completar satisfactoriamente el minijuego y además **se muestran paralelamente dos códigos, habiendo uno que presenta errores de concepto y otro similar que no los tiene**. De esta forma, se **reduce la carga cognitiva** asociada al problema al **enfocar la tarea a un objetivo específico**, lo que permite que los estudiantes aclaren sus dudas al comparar ambos códigos y fortaleciendo su aprendizaje.

Las diferentes categorías de enunciados ofrecieron múltiples formas de presentar los errores de concepto en los minijuegos, abordando una variedad de escenarios con los que mantener la atención del jugador. Además, a medida que el jugador progresa, se enfrentará a ejercicios cada vez más complejos, asegurando un **aumento gradual en la dificultad** para mantener un ambiente desafiante.

5.2.2. Validación de la efectividad de *Pyconception*

El método para validar si *Pyconception* logró ayudar a mitigar los *misconceptions* más comunes en el curso fue **comparar el desempeño del grupo experimental y el grupo de control mediante los post test** que se tomaron al final de cada sesión de ayudantía en la que se jugó.

Si bien al inicio del curso (tabla 2) los grupos eran equivalentes, favoreciendo ligeramente al grupo de control, se evidenció un mejor desempeño, con diferencias significativas, del grupo experimental en dos de las unidades donde se hicieron evaluaciones. Tal como se aprecia en la figura 33, **el desempeño obtenido en los post test por el grupo experimental siempre fue mejor que el del grupo de control**, dando cuenta de la efectividad que tuvo el juego en estudiantes que cursan la asignatura de programación. Por lo tanto, se comprobó que ***Pyconception* cumple con su objetivo de mitigar *misconceptions***.

Se destaca el hecho de que **los estudiantes del grupo experimental mostraron resultados más homogéneos** en comparación al grupo de control, lo que se atribuye a la retroalimentación entregada por *Pyconception* mientras jugaban, permitiéndoles corregir rápidamente sus errores.

A partir de los post test realizados, se puede observar que **la UVA 3**, correspondiente a condicionales, **es la unidad en la que se presentaron más errores de concepto**, ya que los resultados del post test respectivo fueron los más bajos para ambos grupos. Sin embargo, esto **se atribuye a la poca experiencia que los estudiantes tenían** al momento de rendir el post test, dado que en las siguientes instancias los resultados tienden a subir.

El mayor limitante para la validación del juego fue la cantidad de estudiantes, la cual variaba entre las diferentes instancias que se hicieron a lo largo del curso en función de la disponibilidad horaria de los estudiantes que participaron, lo que dificultó mantener un volumen constante. Debido a este problema, **la asistencia de estudiantes en el post test de la UVA 5 fue la más baja de todas**, lo que impidió obtener información confiable de esos resultados.

De todas formas, **se puede observar un aumento en la autoeficacia de los estudiantes del grupo experimental**, ya que en el test cualitativo, muchos estudiantes afirmaron ser capaces de realizar varias acciones necesarias para la resolución de problemas. En contraste, **el grupo de control reportó mayores dificultades** en estos aspectos. Se considera que ***Pyconception* es un factor que influyó fuertemente en este contraste**, puesto que al ayudar a corregir rápidamente *misconceptions*, los estudiantes lograron adquirir mayor confianza en sus propias

capacidades.

5.3. Impacto de *Pyconception* en los estudiantes

Desde un principio se pretendía que los estudiantes pudieran **jugar *Pyconception* durante el bloque destinado para ayudantía**, ya que al ser un bloque establecido en el horario del curso, **se podría contar con una mayor participación** que si se dejaba para una instancia extracurricular. Además, al estar la ayudantía separada de los bloques de cátedra, también **se evitaría interrumpir las labores de docencia del profesor**.

Esta decisión demostró ser acertada, puesto que **participaron alrededor de 20 estudiantes en cada instancia**, tanto en el grupo experimental como en el grupo de control, **salvo por el post test tomado para la UVA 5**, debido a la disponibilidad horaria de los participantes.

Durante la primera sesión de juego con el grupo experimental, se pudo observar el **rápido interés hacia *Pyconception***. Los estudiantes lograron comprender el objetivo del juego, mostrando afinidad con los controles y el diseño de los niveles. Los tutoriales también les permitieron aclarar dudas a medida que se introducían nuevas características.

Pyconception fue bien recibido por el grupo experimental, ya que los participantes se entusiasmaban cada vez que había una nueva sesión de juego. **El aspecto más valorado por los estudiantes fue el apartado interactivo**, ya que a medida que iban recolectando las llaves de los niveles, podían comprender cómo funciona el código en *Python*. **La cantidad de niveles propuestos logró mantener la atención de los estudiantes en sesiones de 30 minutos**. Incluso habían estudiantes que llegaban antes de la hora de ayudantía para poder jugar más tiempo. Por lo tanto, se puede afirmar que ***Pyconception* también motivó a los estudiantes a adquirir el contenido del curso**.

Por otro lado, el test cualitativo que se tomó al grupo experimental contenía preguntas adicionales para que los participantes pudieran evaluar su experiencia con el juego. Ahí, los estudiantes coincidieron en que las sesiones en las que jugaron fueron provechosas y que les serían de utilidad para el primer certamen del curso. Además, comentaron que el juego es una **herramienta valiosa para quienes están empezando a programar**, resaltando lo entretenido que les pareció y cómo les ayudó a mejorar.

5.4. Trabajo futuro

A partir de las experiencias realizadas en este trabajo y de los resultados obtenidos, se ha determinado que ***Pyconception* cumple con los objetivos establecidos al inicio de este documento**. Sin embargo, también se identificaron algunos aspectos que pueden mejorarse, para favorecer un mejor desarrollo del pensamiento computacional a lo largo del curso.

5.4.1. Nuevos minijuegos

Si bien la mecánica de los minijuegos cumplió con su función de ayudar a mitigar errores de concepto, puede volverse monótona. Por lo tanto, se propone crear **nuevos tipos de minijuegos que varíen el nivel de andamiaje** y presenten un desafío diferente a los jugadores.

Un ejemplo sería **completar un espacio en blanco en algún código**, disponiendo de un menú desplegable que contenga varias respuestas posibles, dónde solo una sea la correcta y las demás presenten *misconceptions*. También se puede **variar esta idea en niveles más avanzados**, cambiando el menú desplegable por texto que el jugador debe ingresar usando el teclado.

5.4.2. Abordar más unidades

***Pyconception* cubre las unidades que se evalúan en el primer certamen**, desde la unidad de secuenciales hasta la unidad de funciones. Luego de la última sesión de juego, los estudiantes volvieron a la modalidad tradicional de ayudantías, mencionando varias veces que les habría gustado poder jugar más niveles en los que se abordaran los demás contenidos del curso.

Las unidades que no fueron abordadas en ente trabajo corresponden a colecciones, tales como listas y diccionarios. Para crear nuevos niveles, sería necesario crear situaciones en las que se puedan poner a prueba las operaciones más comunes con estas estructuras, como agregar en una posición particular, trabajar con listas anidadas, etc.

5.4.3. Enfoque cooperativo

En las diferentes experiencias se pudo observar que los estudiantes a menudo conversaban entre ellos sobre cómo superar un nivel. Este comportamiento puede aprovecharse para crear **nuevas mecánicas orientadas a la cooperación entre pares**.

La mejor opción para esto sería **implementar un modo multijugador**, ya sea de forma local o a través de Internet, donde las acciones de un jugador afecten a sus compañeros. Esto permitiría crear situaciones en las que un jugador deba realizar ciertas acciones para que los demás puedan avanzar, y luego ellos hagan lo mismo para que todos recojan las llaves necesarias. También se podría incluir un **sistema de intercambio de llaves**, permitiendo que un jugador transfiera llaves innecesarias a otro que sí las requiera.

ANEXOS

A. Post test

En este anexo se incluyen todos los post test que se realizaron en las UVA 2 a 6. En cada test se pide el nombre del estudiante, para así poder distinguir a aquellos pertenecientes a la carrera de Ingeniería Civil Mecánica; además de 3 preguntas de opción múltiple, con 5 alternativas donde solo una es la correcta. En este anexo, **las respuestas correctas se encuentran en negrita**.

A.1. Diagnóstico post ayudantía UVA 2

1. Considere el siguiente programa en Python:

```
a = 15
b = a
b += 7
print(a)
```

   El valor que imprime es:

   ***a*) 15**

   b) 17

   c) 8

   d) 20

   e) 22

2. Considere el siguiente programa en Python:

```
a = 10
b = a
a = 20
print(b)
```

El valor que imprime es:

a) 20

b) 15

***c*) 10**

d) 30

e) 35

3. Considere el siguiente programa en Python:

```
a = 3
# La respuesta va aquí
c = a + b
print(c)
```

¿Qué instrucción debe insertarse en la línea comentada para que, al finalizar el código, el valor que se imprime sea 10?

a) `-10 = b`

b) `b = 10`

***c*) `b = 7`**

d) `-13 = b`

e) `b = -13`

A.2. Diagnóstico post ayudantía UVA 3

1. Considere el siguiente programa en Python:

```
a = 5
b = 0
c = 0
if a > 0:
    b = 10
else:
    c = 10
```

¿Cuáles son los valores que tienen las variables “a”, “b” y “c” al finalizar el código?

a) a = 5, b = 10 y c = 10.

b) a = 0, b = 5 y c = 10.

c) a = 10, b = 10 y c = 10.

***d*) a = 5, b = 10 y c = 0.**

e) a = 5, b = 10 y c = 5.

2. Considere el siguiente programa en Python:

```
x = 10
y = 5
z = 0
if x > 15:
    if y > 10:
        print(y)
    else:
        print(z)
```

¿Qué acciones realiza el código?

a) El código imprime 5.

b) El código imprime 5 y 10, en líneas separadas.

c) El código imprime 10 y 5, en líneas separadas.

d) El código imprime 0.

***e*) El código se ejecuta sin imprimir nada.**

3. Considere el siguiente programa en Python:

```
a = 10
b = 15
c = 20
if a > 5:
    if b > 10 and c > 20:
        c = 15
else:
    a += 10
suma = a + b + c
print(suma)
```

¿Cuál es el valor de la variable “suma” al finalizar el código?

a) 55

***b*) 45**

c) 35

d) 40

e) 50

A.3. Diagnóstico post ayudantía UVA 4

1. Considere el siguiente programa en Python:

```
x = 0
while x < 5:
    x += 1
print(x)
```

El valor que se imprime es:

a) 0

b) 1

c) 4

***d*) 5**

e) 6

2. Considere el siguiente programa en Python:

```
a = 0
while a < 6:
    if a >= 3:
        print(a)
    a += 1
```

Los valores que se imprimen son:

a) 0, 1 y 2

b) 0, 1, 2 y 3

***c*) 3, 4 y 5**

d) 3, 4, 5 y 6

e) 0, 1, 2, 3, 4, 5 y 6

3. Considere el siguiente programa en Python:

```
n = 5
suma = 0
# La respuesta va aquí
    suma += n
print(suma)
```

¿Qué instrucción debe insertarse en la línea comentada para que, al finalizar el código, el valor que se imprime sea 20?

a) `while suma < 15:`

***b*) `while suma < 20:`**

c) `while suma <= 20:`

d) `while suma > 20:`

e) `while suma >= 20:`

A.4. Diagnóstico post ayudantía UVA 5

1. Considere el siguiente programa en Python:

```
x = "10234"
print(x[1])
```

El valor que se imprime es:

***a*) “0”**

b) “1”

c) “4”

d) “5”

e) “6”

2. Considere el siguiente programa en Python:

```
mensaje = ""
s = "unidad"
s1 = "Unidad 5: Strings"
s2 = "Quiero 3 unidades"
if s in s1:
    mensaje = "primero"
elif s not in s2:
    mensaje = "segundo"
else:
    mensaje = "tercero"
print(mensaje)
```

Los valores que se imprimen son:

a) “”

b) “primero”

c) “segundo”

***d*) “tercero”**

e) “ ”

3. Considere el siguiente programa en Python:

```
a = "programación es una asignatura de la USM"
b = "la USM tiene un curso de programación"
# La respuesta va aquí
    print("iguales")
```

¿Qué instrucción debe insertarse en la línea comentada para que, al finalizar el código, se ejecute el print?

a) `if a[1:12] == b[-12:]:`

b) `if a[0:12] == b[-12:-1]:`

c) `if a[1:13] == b[-13:-1]:`

d) `if a[0:11] == b[-11:-1]:`

e) **`if a[0:12] == b[-12:]:`**

A.5. Diagnóstico post ayudantía UVA *6*

1. Considere el siguiente programa en Python:

```
def func1(n):
    n = str(n)
    s = 0
    i = 0
    while i < len(n):
        s += int(n[i])
        i += 1
    return s
```

¿Cuál de los siguientes llamados a la función entrega el resultado mayor?

a) `func1(50)`

b) `func1(33)`

c) `func1(34)`

d) **`func1(45)`**

e) `func1(123)`

2. Considere el siguiente programa en Python:

```
def func2(a, b, c):
    if len(a) >len(b):
        if len(a) == len(c):
            return 2 * len(a)
        return len(a) + len(c)
    elif len(b) <len(c):
        return len(b) - len(a)
    else:
        return len(c) - len(a)

x = "madera"
y = "acero"
z = "titanio"
print(func2(y, x, z))
```

El valor que se imprime es:

a) -1

***b*) 1**

c) 0

d) 13

e) 12

3. Considere el siguiente programa en Python:

```
# La respuesta va aquí
    i = 0
    t = 0
    while i <b:
        t += a
        i += 1
    return t
```

¿Qué instrucción debe insertarse en la línea comentada para definir correctamente "func3"?

a) `define func3(i, t):`

b) `def func3(i):`

***c*) `def func3(a, b):`**

d) `def func3(a = input(), b=input()):`

e) `define func3():`

B. Test cualitativo

En esta sección se incluye el test cualitativo que se les tomó a ambos grupos luego de realizar todas las sesiones de *Pyconception*. Este diagnóstico cuenta con 10 preguntas dirigidas a ambos grupos, sin embargo, el grupo experimental tiene 3 preguntas adicionales para que puedan evaluar su experiencia con *Pyconception*. Para el test se utilizó la escala de Likert. Cada pregunta cuenta con 5 posibles respuestas que van desde “muy en desacuerdo” hasta “muy de acuerdo”; a excepción de la última, que es de desarrollo.

1. Puedo idear una estrategia adecuada para resolver un ejercicio de programación determinado en poco tiempo.
2. Puedo escribir un programa largo y complejo en Python para resolver cualquier problema dado, siempre y cuando las especificaciones estén claramente definidas.
3. Puedo planificar las etapas y variables necesarias para resolver un ejercicio de programación.
4. Puedo hacer uso de una función pre-escrita, claramente definida.
5. Puedo administrar mi tiempo eficientemente si tuviera una fecha límite en una tarea de programación.
6. Podría encontrar formas de superar un problema si me quedara atascado en un punto mientras resuelvo un ejercicio de programación.
7. Soy capaz de explicar el funcionamiento de mi programa paso a paso.
8. Puedo depurar (corregir todos los errores) un programa largo y complejo que he escrito y hacerlo funcionar.
9. Soy capaz de interpretar la retroalimentación que me dan sobre mis programas.
10. Puedo corregir errores en mis programas si recibo retroalimentación.
11. Disfruté las sesiones de ayudantía en las que jugábamos Pyconception.
12. Creo que los contenidos vistos en Pyconception me serán de utilidad para el próximo certamen.
13. ¿Hay algo más que quieras comentar respecto a tu experiencia en Pyconception?

REFERENCIAS BIBLIOGRÁFICAS

[Bandura, 1986] Bandura, A. (1986). *Social Foundations of Thought and Action*. Prentice Hall.

[Bebras, 2004] Bebras (2004). Bebras - informatics contests for all. `https://www.bebras.org/`. Accedido el 10 de noviembre de 2023 a las 14:30.

[Bebras, s f] Bebras (s. f.). Bebras task examples. `https://www.bebras.org/examples.html`. Accedido el 10 de noviembre de 2023 a las 14:33.

[Bonar y Soloway, 1985] Bonar, J. y Soloway, E. (1985). Preprogramming knowledge: A major source of misconceptions in novice programmers. *Hum.-Comput. Interact.*, 1(2):133–161.

[Caceffo *et al.*, 2018] Caceffo, R., Gama, G., Benatti, R., Aparecida, T., Caldas, T., y Azevedo, R. (2018). A concept inventory for cs1 introductory programming courses in c.

[Campus, 2023] Campus, C. (2023). ¿qué es un motor de juegos? `https://creativecampus.universidadeuropea.com/blog/motor-juegos/`. Accedido el 3 de julio de 2024.

[CodeCombat, 2013] CodeCombat (2013). Codecombat - coding games to learn python and javascript. `https://codecombat.com/`. Página web de CodeCombat. Accedido el 5 de octubre de 2023 a las 19:37.

[Czikszentmihalyi, 1990] Czikszentmihalyi, M. (1990). Flow: The psychology of optimal experience.

[Dodobara, 2005] Dodobara, F. R. (2005). Influencia de la autoeficacia en el ámbito académico. *Revista Digital de Investigacion en Docencia Universitaria*, pp. 1–16.

[Fabila Echauri *et al.*, 2012] Fabila Echauri, A. M., Minami, H., e Izquierdo Sandoval, M. J. (2012). La escala de likert en la evaluación docente: Acercamiento a sus características y principios metodológicos. *Perspectivas docentes*, 1(50):31–40.

[García, 2006] García, J. M. B. (2006). El mercado del videojuego: unas cifras. *ICONO 14, Revista de comunicación y tecnologías emergentes*, 4(1):1–11.

[Gari *et al.*, 2018] Gari, M. R. N., Walia, G. S., y Radermacher, A. D. (2018). Gamification in computer science education: A systematic literature review. En *2018 ASEE Annual Conference & Exposition*, número 10.18260/1-2–30554. ASEE Conferences. `https://peer.asee.org/30554`.

[Godot Engine, 2024] Godot Engine (2024). Godot engine. `https://godotengine.org`. Accedido el 3 de julio de 2024.

[Kaczmarczyk *et al.*, 2010] Kaczmarczyk, L. C., Petrick, E. R., East, J. P., y Herman, G. L. (2010). Identifying student misconceptions of programming. En *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, p. 107–111, New York, NY, USA. Association for Computing Machinery.

[Kazimoglu, 2020] Kazimoglu, C. (2020). Enhancing confidence in using computational thinking skills via playing a serious game: A case study to increase motivation in learning computer programming. *IEEE Access*, 8:221831–221851.

[Kodea, sf] Kodea, F. (s.f.). La hora del código - tutoriales. `https://horadelcodigo.cl/tutoriales/`. Página web de La Hora del Código. Accedido el 5 de octubre de 2023 a las 20:04.

[Lara Mira, 2022] Lara Mira, J. A. (2022). Diseño y desarrollo de un juego serio para aprender pensamiento computacional aplicando gamificación.

[López Reguera *et al.*, 2011] López Reguera, J., Hernández Rivas, C., y Farran Leiva, Y. (2011). Una plataforma de evaluación automática con una metodología efectiva para la enseñanza/aprendizaje en programación de computadores.

[Marc, 2001] Marc, P. (2001). Digital natives, digital immigrants. *On the horizon*, 9(5):1–6.

[Meza *et al.*, 2024] Meza, F., Vásquez, A., y San Martín, D. (2024). Validation of a bebras-based test to assess computational thinking abilities in first-year college students. En *To appear in Proceedings of the 55th ACM Technical Symposium on Computer Science Education – SIGCSE 24*, New York, NY, USA. Association for Computing Machinery.

[MINEDUC, 2017] MINEDUC (2017). Mi taller digital de programación 2017. Accedido el 5 de octubre de 2023 a las 18:45.

[MINEDUC, 2018] MINEDUC (2018). Plan de estudio 2018.

[MINEDUC, s f] MINEDUC (s. f.). Qué es el plan de lenguajes digitales. `https://sitios.mineduc.cl/lenguajesdigitales/que-es-el-plan.html`. Ministerio de Educación, Gobierno de Chile. Accedido el 8 de noviembre de 2023 a las 16:27.

[Mooney y Lockwood, 2020] Mooney, A. y Lockwood, J. (2020). The analysis of a novel computational thinking test in a first year undergraduate computer science course. *The All Ireland Journal of Teaching and Learning in Higher Education*, 12(1):1–27.

[Muñoz, 2023] Muñoz, S. (2023). Videojuego cooperativo para el desarrollo de pensamiento computacional mediante estrategias con baja carga cognitiva.

[Oliveira *et al.*, 2021] Oliveira, A. L. S., Andrade, W. L., Guerrero, D. D. S., y Melo, M. R. A. (2021). How do bebras tasks explore algorithmic thinking skill in a computational thinking contest? En *2021 IEEE Frontiers in Education Conference (FIE)*, pp. 1–7.

[Pérez-Pueyo *et al.*, 2019] Pérez-Pueyo, Á., Alcalá, D. H., Gutiérrez-García, C., y Garijo, A. H. (2019). Andamiaje y evaluación formativa: Dos caras de la misma moneda. *Revista Infancia, Educación y Aprendizaje*, 5(2):559–565.

[Qian y Lehman, 2017] Qian, Y. y Lehman, J. (2017). Student's misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–24.

[Santander, 2023] Santander, B. (Fecha de acceso: 5 de octubre 2023). Gamificación en el aula.

[Sorva, 2012] Sorva, J. (2012). *Visual program simulation in introductory programming education*. Aalto Univ. School of Science.

[Sweller *et al.*, 1998] Sweller, J., Van Merrienboer, J. J., y Paas, F. G. (1998). Cognitive architecture and instructional design. *Educational psychology review*, 10:251–296.

[Tang *et al.*, 2020] Tang, X., Yin, Y., Lin, Q., Hadad, R., y Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, 148:103798.

[Torrent, 2019] Torrent, C. (2019). Programación computacional en escuelas. Tesis de máster.

[Unity Technologies, 2024] Unity Technologies (2024). Unity. `https://unity.com/es`. Accedido el 3 de julio de 2024.

[Unreal Engine, 2024] Unreal Engine (2024). Unreal engine 5. `https://www.unrealengine.com/en-US/unreal-engine-5`. Accedido el 3 de julio de 2024.

[Vásquez Pinto, 2020] Vásquez Pinto, F. J. (2020). Impacto del uso de un lenguaje basado en bloques en el desarrollo de habilidades de programación.

[Wing, 2011] Wing, J. (2011). Research notebook: Computational thinking—what and why. *The link magazine*, 6:20–23.

[Wing, 2006] Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35.

[Wing, 2008] Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3717–3725.