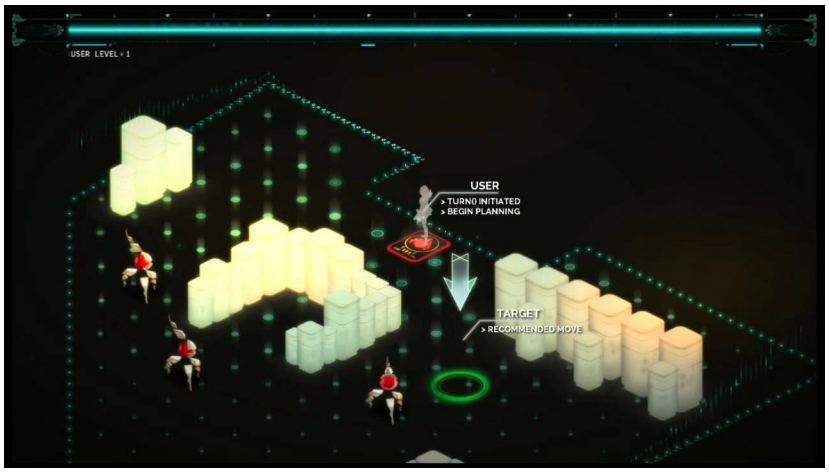
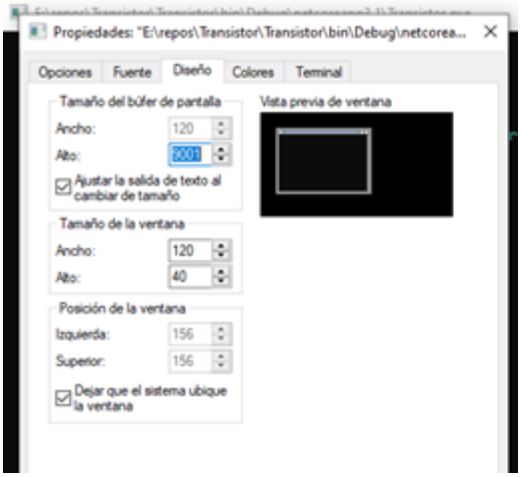
Transistor Proyecto Final FP2 - Marta Croche Trigo



Instrucciones de funcionamiento del juego en consola

El juego requiere que las propiedades de la consola estén ajustadas a una altura mínima de 40 y un ancho mínimo de 110.

Para que funcione correctamente el ejecutable, dentro del bin/Debug/netcoreapp3.1 deben estar los archivos txt de niveles (TransistorX.txt), de la explicación de los controles (Controls.txt), y de los créditos del juego (Credits.txt). Además, hace falta la carpeta fx con los archivos wavs que están dentro.



Mecánicas del jugador

Cada nivel consta de un escenario con varios enemigos y el jugador tendrá dos formas de enfrentarse a ellos:

1. **Ataques en tiempo real**: Al presionar la tecla numérica correspondiente a cada ataque *(Crash: 1, Breach: 2, Ping: 3, Load: 4)*, el jugador lo ejecutará inmediatamente, pero estos tendrán un tiempo de recarga en este modo que impedirá que se repita el ataque justo después.

   Al presionar la barra espaciadora pasará a panear sus ataques y movimientos con Turn si la barra de este está completa. Durante esta fase, Turn se recuperará gradualmente si está por debajo del máximo.

2. **Ataques planeados con Turn:** El jugador tiene disponibles cuatro ataques diferentes: Crash, Breach, Load y Ping.

   Turn() es un ataque que detiene los movimientos de los enemigos y del jugador, mientras este planea sus próximos ataques.

   1. Durante Turn, el jugador selecciona qué movimientos realizar en el escenario (mediante los cursores) y qué ataques usar contra los enemigos en el

escenario, con las teclas numéricas de cada uno, que le mostrarán la dirección y posición de estos si los planea; al presionar Enter, quedarán planeados.

2. Planear cada uno de los ataques gastará una cantidad determinada de la barra de Turn, que representa la “memoria” que requiere cada ataque. Asimismo, moverse en el escenario gastará una cantidad de Turn por casilla de distancia.
3. Con la tecla de Retroceso, se deshace un movimiento o ataque ya planeado de Turn, y se recupera su cantidad correspondiente de la barra.
4. Al presionar de nuevo la barra espaciadora, el personaje del jugador ejecutará automáticamente los movimientos y ataques uno por uno, estando los enemigos y proyectiles enemigos aún paralizados. Tras terminarlos, volverá al combate en tiempo real con los enemigos, y no podrá usar Turn hasta que la barra vuelva a estar completa.

- **Ataques:**
    - *Crash:* Necesita que el jugador esté en una casilla adyacente al enemigo y hace poco daño, gasta un quinto de Turn.
    - *Breach*: Puede disparar a cualquier distancia y daña a todo en su línea de ataque, destruyendo muros a su paso. Hace mayor daño, pero gasta algo más de un tercio de Turn.
    - *Load:* Coloca una mina, que podrá ser activada por cualquier otro ataque, y al explotar hará daño en un área en forma de rombo, gasta la mitad de Turn.
    - *Ping:* Dispara un proyectil que hace poco daño al enemigo al que alcanza, gasta un décimo de Turn.

3. **<u>Vidas del jugador:</u>** Cada vez que la vida del jugador llegue a 0, volverá a recuperar toda su vida, perdiendo la capacidad de usar uno de sus ataques, seleccionado por el juego de forma aleatoria, hasta el siguiente nivel. Esto se muestra al jugador haciendo que la casilla de ese ataque se vuelva gris.

    El jugador perderá la partida cuando todos sus ataques estén desactivados.

    Al comenzar un nuevo nivel, el jugador recuperará toda su vida y sus ataques.

<u>Mecánicas de los enemigos</u>

_____El juego tiene 4 tipos de enemigos que cuentan con su propio comportamiento.

- *Creep:* Ataca de forma continua con un láser de proyectiles cuando está en línea recta con el jugador, no puede atacar mientras se mueve.
- *Snapshot*: Dispara proyectiles al jugador, ataca desde lejos y trata de alejarse si no está a una distancia mínima del jugador. Hasta que encuentra al jugador en su línea de visión, trata de seguir su misma dirección, cambiándola cada poco tiempo, para no ir directamente hacia él, como el resto de enemigos.
- *Fetch:* Ataca físicamente al hacer contacto, persigue al jugador y se queda parado durante un tiempo una vez consigue hacer daño al jugador.
- *Jerk:* Ataca en área en forma de rombo por cada movimiento.

Mecánicas de escenario

Los escenarios contarán con dos tipos de muros: de límite de escenario, y de interior. Los de interior podrán ser destruidos con Load y Breach.

Acciones extra

Con Escape, se puede salir del juego, y con la tecla M, se vuelve al menú del juego.

Estructura de clases

class Transistor → Esta clase se encarga del estado de Turn (normal, plan y run) y del bucle principal del juego, Run()

class Program → Esta clase se encarga de llamar a la ejecución del juego. Crea el objeto Transistor game, y llama a game.Run()

class Battlefield → Esta clase se encarga de controlar el estado actual del nivel del juego y de mostrarlo en pantalla según el momento de la partida.

class Character → Esta clase define algunas acciones, variables y propiedades comunes para el jugador y los enemigos.

- **class Player: Character** → Esta clase controla las acciones del jugador (movimientos y ataques, que dependen del estado del juego) y sus atributos, así como las acciones a ejecutar con Turn.

- **class Enemy: Character** → Esta clase define el movimiento básico y algunas variables y propiedades comunes de los enemigos.

  Estas clases controlan el movimiento y los ataques específicos de cada tipo de enemigo:

  - ➔ **class Creep: Enemy**
  - ➔ **class Snapshot: Enemy**
  - ➔ **class Jerk: Enemy**
  - ➔ **class Fetch: Enemy**

  class EnemyList → Lista enlazada de los enemigos.

class Projectile → Esta clase contiene los atributos y métodos comunes para la mayoría de proyectiles.

Estas clases contienen el comportamiento y atributos específicos de cada tipo de proyectil:

- ➔ **class Laser: Projectile**
- ➔ **class Shot: Projectile**

➔ **class Beam: Projectile**
➔ **class Bullet: Projectile**
➔ **class Load: Projectile**

class ProjectileList → Lista enlazada de proyectiles.

class Coor → Esta clase contiene los métodos del tipo de datos Coor.

class CaptionDisplay → Esta clase guarda la representación de cada elemento del juego y la escribe en una leyenda a la derecha del nivel, así como el número del nivel en el que se encuentra el jugador en ese momento.

class TurnDisplay → Esta clase pinta en pantalla las barras de Turn y de vida del jugador, así como los ataques disponibles en su color correspondiente (o en gris si han quedado deshabilitados), con su número de tecla indicado.

class Menu → Esta clase se encarga de mostrar y ejecutar el menú inicial del juego, que tiene opción de continuar una partida guardada (si hay algún usuario ya registrado), iniciar una partida nueva, mostrar una explicación de los controles, mostrar los créditos del juego y salir del juego.

class SoundFx → Tiene guardados los archivos de efectos de sonido y la música del juego. Estos primeros tienen todos sus propios objetos para que funcionen algo más rápidamente durante los niveles, y la música se leerá del archivo cada vez que tenga que sonar (en el menú y al finalizar el juego).

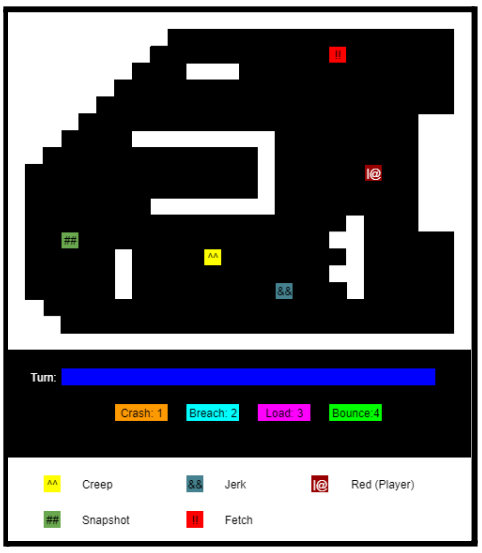
Representación

Los niveles estarán en un archivo txt que utilizará una codificación de números y letras para los niveles.
En el que las tiles son: 0 → Empty, 1 → Wall, 2 → borderWall
Los enemigos son: C → Creep, S → Snapshot, J → Jerk, F → Fetch
Y R → Red (jugador)

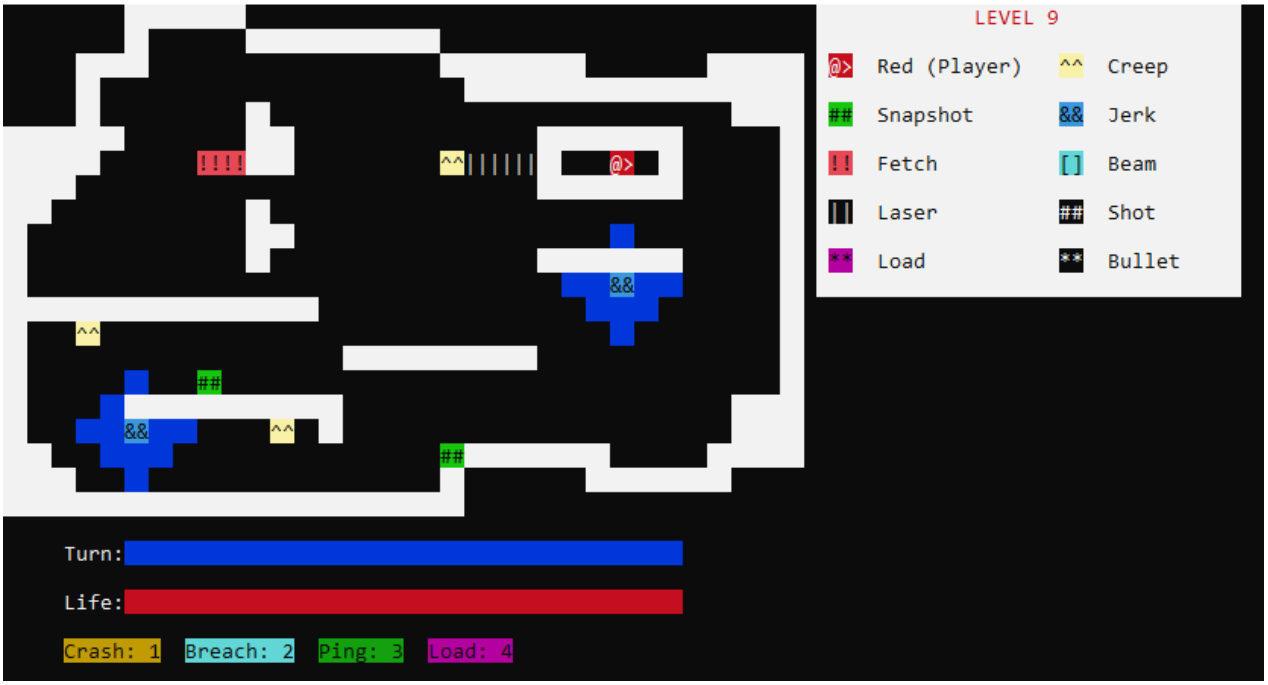
Ejemplo de diseño de un archivo de nivel

```
22222222222222222222222222
22222222200000000000000002
22222222000000000F0000002
22222220001110000000000002
22222200000000000000000002
22222000000000000000000222
22220000111111110000000222
22200000000000010000000222
22000000000000010000R00222
20000000000000010000000222
20000000111111110000000222
20000000000000000001000222
200S0000000000000011000002
20000010000C00000001000002
20000010000000000011000002
22000010000000J0001000002
22200000000000000000000002
22222222222222222222222222
```

Ejemplo de representación en consola de un nivel



Representación real del juego en consola.



Bibliografía y Créditos

- El concepto original del juego es de Supergiant Games: https://www.supergiantgames.com/games/transistor/
- La herencia de clases la he aprendido de Codigofuentenet: https://codigofuentenet.wordpress.com/2012/07/17/polimorfismo-en-c/
- La herencia de constructores la he sacado del siguiente hilo de Stack Overflow: https://stackoverflow.com/questions/3873343/can-i-inherit-constructors
- El funcionamiento de los sonidos en c# lo he aprendido de las siguientes páginas: http://csharphelper.com/blog/2016/08/play-wav-files-in-c/ https://docs.microsoft.com/es-es/dotnet/api/system.media.soundplayer.play?view=net-5.0 https://www.reddit.com/r/csharp/comments/cvcl3z/cant_use_systemmedia_and_cant_find_correct_dll_to/
- Los efectos de sonido del juego son de MixKit: https://mixkit.co/free-sound-effects/
- La canción que suena en el menú del juego es Guardian of Power de Grégoire Lourme: https://www.jamendo.com/album/200402/guardian-of-power?language=en
- La canción que suena al finalizar el juego es la versión para testing del Cinematic Emotional Trailer de Marc Corominas Pujadó. https://licensing.jamendo.com/en/track/1820640/cinematic-emotional-trailer