ExiteBike

LABORATORIO 2

Diego Dave, Federico Garcia, Yenniffer Herrera

VIDEO JUEGOS 2D | PROF.: ANDRÉS PASTORINI

MELONJS Y TILED MAP EDITOR

¿QUE ES?

MelonJs es un motor javascript para desarrollar juegos HTML5, es especifiamente para sprites 2D compatible con todos los navegadores. Con la cual se puede trabajar animaciones, fisica de coliciones, multiples pistas de audio, loader customizable y lo mas destacado es que posee integracion con Tiled Map Editor (a partir de la version 0.7)

Tiled Map Editor es una herramienta para realizar mapas de tiled, cuenta con un formato basado en XML llamado TMX, soporta tanto mapas ortogonales como isometricos, tiene soporte para manejar los pixeles para una mayor precisión permitiendo no tener que basarse unicamente en los tileds.

CARACTERISTICAS

MELONJS

- Es una biblioteca independiente que no depende de nada más que de un browser que soporte HTML5
- Compatible con todos los navegadores (Chrome, safari, opera Firefox, IE)
- Soporta varios canales de audio
- Maneja mecanismos de colisión y física básica para asegurar bajo requerimiento de CPU
- Efectos de transacción
- Conjunto básico de entidades.
- Gestión básica de animación
- Gestor de estados
- Sistema de fuentes de mapa de bits
- Emula mouse y dispositivos táctiles
- Mapa de Tiled (transparencias, múltiples capas, perspectivas etc.)

TILED MAP

- Mapas ortogonales e isométricos
- Formato de archivos XML
- Soporte para múltiples capas tanto de tileds como de objetos
- Soporte para múltiples tileds
- Propiedades altamente personalizables
- Soporta color y transparencias
- Tamaño de tileds y mapas totalmente personalizados

INSTALACIÓN

Para instalar MelonJs es Preferible instalar

- **Node.js npm (**Node Packaged Module**)** que es una plataforma construida en tiempo de ejecución de javascript para facilitar la creación de aplicaciones de red rápidas y escalables**,** ideal para aplicaciones en tiempo real de datos intensivos que se ejecutan a través de dispositivos distribuidos.
- **Node Packaged Module** este es necesario para automatizar y minimizar la compilación
    - **Entonces en command prompt**
        - **$ npm install -g grunt-cli**
- **melonJs**
    - **Tambien en command prompt**
        - **$ cd melonJS**
        - **$ grunt**
- **Map Tiled Editor** se descarga y ejecuta el instalador.

http://www.mapeditor.org/download.html

http://nodejs.org/

https://npmjs.org/

http://gruntjs.com/

http://melonjs.org/download.html

INTRODUCCION AL DESARROLLO

MELON.JS

COMPONENTES PRINCIPALES

- **me**: melon engine, indica el uso del motor de melonjs.
    - **sys**: Para setear variables globales del juego. Ej: fps, escalas, gravedad.
        - me.sys.dirtyRegion = true;
    - **video**: Para setar funcionalidades de video.
        - me.video.init('jsapp', 1024,192, true))
    - **loader**: Para iniciar los elementos del juego.
        - me.loader.onload = this.loaded.bind(this);
        - me.loader.preload(g_resources);
    - **state**: Para asignarle los diferentes estados que tendra el juego.
        - me.state.change(me.state.LOADING);
        - me.state.set(me.state.MENU, new PantallaInicial());
        - me.state.set(me.state.PLAY, new PantallaJuego());
    - **entityPool:** Para asignar las entidades. Ej: Los jugadores.
        - me.entityPool.add("jugador", PlayerEntity);
        - me.entityPool.add("rival", RivalEntity);
        - me.entityPool.add("rampa", RampaEntity);
        - me.entityPool.add("llegada", LlegadaEntity);
    - **input**: Para asignarle las funcionalidades
        - me.input.bindKey(me.input.KEY.LEFT, "left");
        - me.input.bindKey(me.input.KEY.RIGHT, "right");
        - me.input.bindKey(me.input.KEY.UP, "jump", true);
- **ObjectEntity**: Setea las propiedades de la entidad definido en el tiled con Tiled Map Editor.
    - **parent:** Llama al constructor

- this.parent(x, y , settings);

- **setVelocity**: Setea la velocidad de caminar y saltar.
  - this.setVelocity(9, 10);
- **setFriccion**: Le da fricción a la entidad
  - this.setFriction(0.4);
- **updateColRect**: Asigna el tamaño de la entidad.
  - this.updateColRect(32,32, -1,0);
- **updateMovement**: Actualiza el movimiento de la entidad.
  - this.updateMovement();
- **doWalk:** Para asignarle el movimiento a la entidad
  - isKeyPressed('left')
    - this.doWalk(true);
  - isKeyPressed('right')
    - this.doWalk(false);
- **doJump**: Para asignarle el salto a la entidad
  - isKeyPressed('jump'))
    - this.doJump();
- **updateColRect**: Comprueba la colision de la entidad.
  - me.game.collide(this);

TILED MAP EDITOR

COMPONENTES PRINCIPALES

- **Mapa:** Donde se ubicaran los objetos y conjuntos de patrones.
- **Objetos:** Son los que tendrán un comportamiento determinado y permitirán al usuario interactuar con ellos.
- **Capas:** Permiten trabajar de una manera mas cómoda separando los diferentes tipos de elementos.
- **Patrones:** Son los utilizados para crear el ambiente del juego, a los cuales es posible asignar un comportamiento que interactúan con los elementos u objetos del juego.
- **Imágenes:** Permiten generar elementos estáticos sin comportamiento dentro del juego.

OBJETOS

Se utilizan para asignar un comportamiento determinado.

En nuestro juego de ejemplo son, El jugador y el rival

Es necesario asignarle un conjunto de atributos para que nuestro framework los pueda ejecutar.

- **Imagen:** Para asignarle el nombre de la imagen que corresponde a ese elemento
- **Spriteheight:** Para asignarle el alto de la imagen
- **Spritewidht:** Para asignarle el ancho de la imagen

PATRONES

Son utilizados para crear el ambiente del juego y es posible asignarle un comportamiento determinado.

En nuestro juego de ejemplo son, los bloques solidos sobre los que se moverá y las rampas en las cuales saltara. Estas actúan como elementos de reacción ante las colisiones que los objetos tendrán.

También, como en los objetos, es necesario asignarle un conjunto de atributos para que nuestro framework los pueda ejecutar.

- **type:** Para asignarle el tipo de elemento y el comportamiento al cual el objeto debe reaccionar.
- Este atributo puede tomar los valores:
    - *solid*: Indica el cuadro en el que no le será posible moverse dentro de el.
    - *lslope*: Indica el cuadro con rampa de la izquierda que impulsa el objeto.

- *rslope*: Indica el cuadro con rampa de la derecha que impulsa el objeto.
- *leader*: Indica el cuadro con escalera que permite subir al objeto.

IMPLEMENTACION DEL JUEGO

Realizamos los Mapas con Tiled Map Editor, en el cual definimos tres capas

1. **Background:** la cual es la pista en si tiled a tiled rampas meta etc…
2. **Entities:** en el cual se encuentran los objetos Jugador, Rival, Rampas y Llegada.
3. **Collision:** en el cual definimos una serie de tiled del tipo Solid y platform.

La entidad Jugador al igual que la entidad Rival está compuesta de los siguientes atributos la imagen = nombre de la imagen, spriteheight = 21 y spritewidth =20

Las entidades Rampa no precisa ninguno de estos atributos dado que la imagen pertenece al background y esta entidad se utiliza únicamente para determinar la zona de colisión.

La estructura de nuestro proyecto en si está formada por un html que contiene únicamente las referencias a los js desarrollados, y la estructura básica de un archivo html.

Además de esto contamos con un Javascript donde definimos las pantallas (pantallas.js), uno para las entidades (entidades.js) y el principal (main.js) en el cual se unifican e inicializan estos antes mencionados.

ENTIDADES.JS

En esta tenemos definida 2 funciones del tipo ObjectEntity y 2 del tipo CollectableEntity estas están relacionadas con las entidades definidas previamente en el mapa de tiled PlayerEntity (Jugador), RivalEntity (Rival), RampaEntity (Rampa), LlegadaEntity (llegada), y una 5ta es para mantener el puntaje del jugador en juego ScoreObjeect del tipo HUD_Item.

Tanto en la entidad Rival como Player definimos la velocidad, la gravedad, la fricción, las acciones al presionar alguna tecla y la posición de cada rampa con la cual puede llegar a colisionar.

Para el control de saltos, se indican las posiciones de cada rampa definida en un arreglo, que es utilizado para el control de la funcionalidad.

Para el control de frenado en los barros, se realiza de la misma forma. Se identifica la posición de cada barro y se implementa la funcionalidad.

Por ultimo en ScoreObjeect se posiciona y dibuja el puntaje obtenido por el jugador al colisionar con cada rampa.

PANTALLA.JS

En este se define la pantalla inicial en la cual se muestra una imagen se escribe el nombre de juego y las acciones posibles de realizar.

Las cuales son elegir jugar con uno o dos jugadores y continuar la ejecución del juego.

También se define la pantalla final en la que muestra el resultado del jugador (rojo) en función de su rival (azul).

Los textos se dibujan mediante BitmapFont el cual despliega el mensaje a partir de las letras definidas en una imagen del tipo png que contiene los dígitos.

MAIN.JS

En este definimos un array de recursos el cual contiene las imágenes, los sonidos y el mapa utilizados en el juego.

Inicializamos las pantallas, el score.

Bindiamos las entidades con los elementos definidos en el mapa y los eventos básicos para el funcionamiento del mismo.

VENTAJAS Y DESVENTAJAS

VENTAJAS

- Software Libre
- Uso de Tiles, buena abstracción
- No requiere nada extra más que el Editor de Tiles para crear los tmx
- Buena documentación
- Fácil de utilizar e integrar luego que se entiende como funciona, ya que las colisiones se chequean en base a los tiles

DESVENTAJAS

- Aún en desarrollo, inestabilidad de funcionalidades
- Se deben coordinar los nombres de las variables y los objetos del archivo de tiles
- Llamar explícitamente las imágenes para que las cargue el navegador
- Al ser javascript, ante un error, no carga y tampoco da información sobre a qué se debe la falla

PROBLEMAS ENCONTRADOS

TILED MAP EDITOR

- **Formato de la capa del Mapa:** En un principio usamos el formato de compresión que venía por defecto en Base64 con compresión *gzip*. Esta característica no mostraba el mapa en el navegador. Al cambiarle a *XML*. Se solucionó el problema.
- **Tamaño del mapa:** Al no definir correctamente el tamaño del mapa con respecto a las dimensiones del conjunto de *tiles*, las entidades no se veían y era porque caían al fondo de mapa, y no encajaban en el lugar deseado.
- **Dimensión de los tiles del mapa:** Al definir los tiles del mapa con un tamaño diferente al de las entidades, no sucedía que el jugador no se movía en relación a las características del mapa.
- **Definición de atributos de los patrones (tile):** Al no definirle un patrón a los tiles que se encontraban en el mapa, nos sucedía que el jugador y el rival no aparecían. Esto se pudo resolver al definirle el tributo de "*solid*" a los tiles inferiores, los cuales oficiaban de piso para las entidades.
- **Definición de atributos de las entidades:** Teníamos que definirle el ancho, alto y el nombre de imagen que le correspondía para que se efectuara el mapeo en el archivo entidades.js. Como estos atributos no estaban definidos, la entidad (el jugador y el rival) no se mostraban en los gráficos y al tener una definición de ancho y alto más grande que la definida en el mapa para cada *tiled*, estas colisionaban mucho antes de lo que correspondía.

MELONJS

- **Posicionamiento:** Para realizar el posicionamiento vertical de las entidades, hubo que recurrir a la implementación manual. No encontramos en el *framework*, alguna operación que implementara las funcionalidades requeridas para el intercambio de carril. Esto requería que la entidad subiera y bajara, y para esto hay una propiedad llamada *doClimb* que permite realizar esas funcionalidades pero siempre y cuando el tile contenga el atributo de tipo *ladder*. Así mismo esta funcionalidad no era adaptable a la realidad del juego realizado.
- **Seguimiento de la entidad en el Mapa:** Uno de los problemas que no pudimos solucionar, fue que la entidad rival (la que no se enfoca en el juego), al tomar la delantera y desaparecer de la pantalla no continua su recorrido y queda esperando el movimiento de la entidad jugador.
- **Coordinación de los tiles de la entidad en el mapa con la entidad en el script:** En el script entidades.js (en el que se encuentran implementadas las entidades del juego), se debe indicar el

tamaño del tile que corresponde a la imagen que lo representa. Para esto debe concordar el tamaño del tile con el tamaño de la imagen y el tamaño para la imagen definido en el mapa del tile. Al no corresponderse los tamaños, la entidad muestra interacciones cambiadas, como ser el choque con una montaña, o el movimiento sobre un carril que no corresponde con el que debería, entre otros.

- **doJump:** En un principio esta propiedad permitía que una entidad al colisionar con otra se le aplicara la funcionalidad del salto. Luego, al cambiar la implementación del juego (por el motivo del cambio del carril), se dejó de utilizar ya que el salto lo hacía por sí mismo al colisionar con otro objeto sin necesidad de usar esta operación. Esta funcionalidad es influida por la gravedad que tenga el juego, ya que si la gravedad era muy alta, el objeto bajaba inmediatamente y no se efectuaba el efecto de un salto.

- **doClimb:** Esta propiedad dio sus problemas al querer implementar la funcionalidad del cambio de carril. Al verificar que esta no era posible por sus propiedades, hubo que dejar de usarla e implementar las funcionalidades que nos eran requeridas con otras operaciones existentes en el *framework*.

- **Detección de las colisiones:** En principio se utilizó la funcionalidad *onCollision* de la entidad, que permitía realizar automáticamente (por como venía definido en el mapa de tiles) la detección de la colisión con algún otro objeto o entidad definidos, pero al cambiar la forma de implementación, se tuvo que detectar en cada posición, la funcionalidad que efectuaría cada entidad y aplicar en ella la funcionalidad requerida.

- **Detección de las motañas:** En principio, como en la detección de colisiones se hacía por medio de la detección de los objetos definidos en el mapa de tiles, no era necesario indicar la posición en la que se encontraban ya que al generarse el mapa este contenía su posición. Al cambiar la implementación por la funcionalidad de intercambio de carriles del jugador, hubo que indicar la posición en las que se encontraban y aplicarles funcionalidades que eran requeridas en sus posiciones. *(var montanas = [1057, 1384, 1816, 1872, 2384, 2440, 2496, 2944, 3036, 4064, 4104, 4144, 4183, 4368, 4688, 4776, 4896, 4984, 5656];)*

- **Contador de puntos:** Para el contador de puntos el problema que hubo fue el de que al no estar definido correctamente el tamaño de los tiles (número o letra) *(this.font = new me.BitmapFont("32x32_font", {x:32}); this.font.set("right", 0.5);)* de las imágenes definidas, en las operaciones no se realizaba correctamente el mapeo y no se realizaba la funcionalidad.

- **Fricción en los barros:** Se implementó al igual que con las montañas, identificando las posiciones de los barros en el mapa y asignando a ellas la funcionalidad de frenado.

- **IA del Rival:** Se implementó al igual que el jugador, pero con la diferencia que el rival se mueve de acuerdo a un patrón aleatorio. Este se realiza obteniendo un numero aleatorio entre 1 y 120 (dos segundos), el cual decrementa hasta 0, momento donde cambio de carril, que se elige de acuerdo a otro random entre 1 y 2.

REFERENCIAS.

- Documentación API melonJS

  http://melonjs.github.io/docs/

- Tutorial de Archivos de Tiles (tmx)

  https://groups.google.com/forum/#!msg/melonjs/hPowvvoPNDU/ql-hz153ZjgJ

- Tiled Map Editor

  http://www.mapeditor.org/

- melonJS

  http://melonjs.org/

- HTML5

  http://www.w3schools.com/html/html5_intro.asp