Changes to TickTick

Wassim Chammat/Corné van Vliet
2981351-6790836

Levels
The levels in the game have been swapped out for 6 custom levels, which show off the new changes to TickTick.

Camera

A camera class has been added to the Game Engine. It is a child class of GameObject in order to give an instance of the class a position that can be traced. An instance of the camera class consists of a rectangle which initially has a (0, 0) offset that can be manipulated through movement of TickTick: while the instance is created in the ExtendedGame class, TickTick's movement can be changed through the Player class. We intentionally did this, because the camera is easily accessible this way (through a property) and now the camera can follow the movement of TickTick easier. If TickTick is facing left and has a X-position of 480 or less (one third of the screen), the camera will start moving left with the same speed as TickTick. The same goes for when TickTick is facing right, but now with an X-position of 960 (two thirds).

Position
The position of SpriteGameObjects is calculated with this formula:
GlobalPosition - ExtendedGame.camera.LocalPosition. So, if the camera would move to the right, the position at which the object is drawn would go left and vice versa.

Boundaries
The camera does not move anymore if the level ends: when the X-position of the camera gets smaller than the grid size (less than 0) a movement boundary is reached. The same goes for the X-position - camera size on the maximum size of the level. This way, the camera will follow Tick Tick when he has reached a certain threshold on the screen but it will stop moving at the edge of the level. The same goes for vertical movement of TickTick.

Manual timer per level
We added a manual way to change the timer for a level: we added an extra line to all the textfiles of the levels from which the levels are read by the level class. In LevelLoading.cs this extra line is read translated to an integer (through int.Parse) that can be used as a timer. This way, every level can have a different time to finish it.

Parallax
Parallax is the phenomenon that causes faraway objects to move past more slowly than closer objects, giving a sense of depth. This was added as a mandatory part of the assignment.

In the SpriteGameObject class, a float variable has been added: scrollFactor. Whenever the camera moves on-screen items along, the distance they move is multiplied by scrollFactor.

Certain basic values have been set:

- The default value for all SpriteGameObject objects is 1. This causes them to scroll by normally, as fast as the camera does.
- For all HUD items, such as the timer, hint window and quit button, scrollFactor has been set to 0. This means their position will never change, which is what a HUD should do: remain in the same position.

For background items (clouds and mountains), a depth variable is used, which is random per cloud or mountain sprite. The scrollFactor for clouds is their depth multiplied by 3, while for mountains, it is their depth times 1.5. This means mountains will scroll by slower than clouds, giving the desired parallax effect in the background.

The sky sprite does not scroll: its scrollFactor is set to 0.

Jumping on rockets
The Rocket class has been expanded.
Whenever TickTick is falling and collides with a rocket (as in: TickTick lands on a rocket), he will jump up based on a set launchSpeed constant. This code functions very similarly to that of the Turtle class. TickTick will not jump as exorbitantly high as he would on a turtle, but the jump is still significant (900, the same value as TickTick's normal jump).

Speed Behavior
A new tile type has been added: the Goo block. Whenever TickTick stands on this block, his movement speed changes to half the usual amount. For it to resemble a real sticky floor, TickTick can jump to stop being slowed down.

Just like all other surface types, there is a Wall variant and a Platform variant, which act as intended. a standingOnGooTile bool is used in the Player class to check for required speed drops. This is checked in Update().

Lives

TickTick no longer dies in one hit: instead, he starts out with 3 lives (and gets 3 after each Reset() call). Whenever he hits an enemy, the TakeDamage() method is called, which decreases his health by 1. If his health reaches zero, Die() is finally called, causing a game over. TickTick is not immune to falling off the screen, however. Life functionality is added to the player class.

Hud

BombTimer.cs has been renamed to Hud.cs, as it now also handles the life display alongside the timer display. The life text blinks red when only 1 life is left, and displays "GAME OVER" in red when TickTick dies or explodes.

Taking damage
Because TickTick can now get hit by an enemy and not die (by taking damage instead), we had to make sure the TakeDamage() method in the Player class does not go on indefinitely for as long as TickTick touches the enemy, as this would cause an instant game over.
The solution to this was a timer that turns on whenever TickTick takes damage. The timer is set to 0.8 seconds and lets TickTick phase through enemies while it is running. This allows for the player to react to the damage, and it creates a form of invincibility time, to make sure the player will not constantly take damage - or even die - beyond their control.
The chosen time is long enough to not cause unwanted interactions (like jumping on the rocket that hit you), but short enough to not allow for game-breaking damage-boosting (which is still possible to a very, very small extent, for those who wish to use it...).

Knockback
While taking damage, TickTick gets "knocked back" based on the way he was facing. Simply taking damage without anything happening is very disorienting to the player, and the knockback acts as a form of feedback to the hit (it can also screw the player up depending on the amount of bottomless pits in the level, so they won't be too reckless with their lives). A vertical and horizontal damageSpeed constant float make up the values for a Jump() method call.

Timer for Sparky
A SetRandomTimer() method has been added to the Sparky class, to better distinguish the random number generation of the timer. It was initially also planned to be used to allow the timer to be reset during Sparky's movement cycle, however, this idea was cast aside.

Pick-Up items

A few collectable items have been added to the game as extra features. These are based on the WaterDrop class and its functionalities. Unlike the water drop, however, they each inflict a special type of effect on TickTick, instead of being a requirement to clear the level.

Some of these items are time-based, and function based on static double timer variables. In the Player class, these timers constantly decrease per second if they are activated.

Health packs
Health packs have been added through a HealthPak class. These red, plus-shaped items are based on the WaterDrop class, and instead of adding to the waterDrops list, they restore lives! You can get more lives than the starting amount, but never more than five (only when the player's health is under 5, does the health pack heal). A small reward for hoarding lives and taking the careful approach.

Shoes
Another item has been added: the shoe. Collecting one of these will make TickTick run much faster and jump higher for 5 seconds. Once collected, a static timer in the Player class starts counting down from 5 seconds until the timer reaches zero. During the countdown, shoeCollected is true, multiplying the horizontal speed by 1.5 in Player.Update(), and the jump height by 1.3 in Player.HandleInput().

Stopwatch
Yet another item, the stopwatch, also functions based on a timer similar to the shoe (this, however, is a separate timer). The stopwatch will freeze the timer for 2 seconds. The code used to count down the level timer is now enclosed by an if-statement, which only allows the timer to count down when the stopwatch effect is not active (if (!Stopwatch.stopwatchCollected)).

Star
A classic item in many platformers, the star is yet another time-based item that makes TickTick immune to damage for 6 seconds. It has a separate timer from the usual damage timer, even though both work based on the canTakeDamage bool. This makes sure that your 6-second invincibility cannot possibly be interrupted by the 0.8-second grace period from touching an enemy.

Corné van Vliet - 6790836