The SpongeBob SquarePants Movie Custom Race Template Documentation

The purpose of this document is to define all of the features included with the TSSM Custom Race Template, as well as including how different elements were implemented, what each assets connection are to each other, and how different elements would have to be theoretically added if desired by the track designer.

The TSSM Custom Race Template is an importable HIP archive created by actualchatterteeth. It was created for level creators that are developing racing levels and would like to be able to have them include different laps and a timer, similar to that of other cart-racing games like Mario Kart and Crash Team Racing. The TSSM Custom Race Template includes all of the assets required to turn a user-created race track into a level able to be raced on like a cart-racer. These assets add the following functionality:

- Lap system that requires players to go through certain triggers in a certain order before the race may end
- Timer system that tracks the lap completion times as well as the race completion time accurate to milliseconds
- On-screen lap counter, timer, and lap times (latter only shows upon lap completion)
- Simple pause menu that allows the player to restart the race
- Simple results screen that allows the player to view their lap and race completion times as well as restart the race
- Disabling of extraneous on-screen HUD icons and traditional pause and upgrade menus

Due to the complexity of how some of these features were added, this document will discuss how a track designer would go about extending the features present in the TSSM CRT. These are following additions that the track designer can make that will be discussed:

- Addition/removal of checkpoint triggers
- Addition/removal of laps
- Removal of milliseconds
- Addition to pause menu
- Modifications/removal of pause menu
- Additions/changes to results screen

IMPORTANT: The TSSM Custom Race Template uses COND assets that utilize TimerMillisecondsLeft, which crashes in-game without the use of an AR Code:

Fixes TimerMillisecondsLeft Crash: 040c2170 4082002c

The Lap System and the Timer System

Before discussing how a track designer would make changes to assets regarding the laps, it’s important to delve into exactly what happens during a race and how the lap system is interconnected with the timer system. At its core, the lap system is fairly simple:

-At the start of the race, the CP_COUNTER is set to 0
-Upon driving through a checkpoint trigger, the CP_COUNTER increases by 1

-Upon driving through the finish line, the CP_COND checks if the CP_COUNTER is equal to the number of checkpoint triggers (the CRT starts with 7 triggers, and so the CP_COND checks if CP_COUNTER is equal to 7 by default)
-If the CP_COND returns true, CP_COUNTER will be reset to 0 and LAP_COUNTER will be incremented
-If the LAP_COUNTER is equal to 4, LAP_COND will return true and the race will conclude. If not, LAP_COUNTER is run through other COND assets to determine the current lap

That is the lap system at its core. There are a few things to note involving the checkpoint system as well that did not need to be mentioned when discussing the lap system:

-Only one checkpoint trigger is enabled at a time, with driving through CP_TRIGGER_(n) disabling itself, incrementing CP_COUNTER and enabling CP_TRIGGER_(n+1)
-The finish line trigger is always active
-Passing through the finish line trigger without going through all of the checkpoints (or the amount that the track designer requires the racer to go through according to CP_COND) will result in CP_COND returning false, causing CP_COUNTER to reset to 0, disabling every single checkpoint trigger, and enabling the first checkpoint triggers

With the current design of the checkpoint triggers, you could theoretically drive backwards through the track, pass through the first checkpoint trigger, then drive forwards through the track normally to complete it. This is a limitation of how the checkpoint system is designed, however I would rather have the system be less complicated and allow for the easier addition and removal of checkpoint triggers by track designers rather than have it be completely unbreakable to the racer but terrible for the track designer to work with. This issue is negated if the track involves a jump of sort that renders the racer unable to backtrack through the course.

As for the timer, or rather timers, there is a fair bit going on under the hood to have everything function as desired. It is important to note the difference between timer types before attempting to add, delete, or modify timers. TIMR assets are float timers that count down from a number and can send an event when the timer is expired. DYNA assets with the type "game_object__RaceTimer" are integer timers that count up from zero and can have a best time set to them. Listed below is every timer that contributes to the timers and times visible to the player throughout the course of the race:

-TIMERACE_TIMER (DYNA: game_object__RaceTimer): This timer starts at the beginning of the race and is stopped at the end of the race. This timer is used to track the race time (not accounting for milliseconds) on the results screen. Once the race is completed, this timer is stopped and the time at the timer is saved as the RaceTimer's Best Time, allowing for easy referral to it in TEXT assets.

-LAP1_TIMER (DYNA: game_object__RaceTimer): This timer starts at the beginning of the race and is stopped at the conclusion of the first lap. This timer is used to track the lap time (not accounting for milliseconds) of the first lap on the HUD as well as on the results screen. Once a lap is completed and, through a series of conditionals, is determined to be the first one completed, this timer will be stopped and LAP2_TIMER will be started. The current time on this timer upon completion of the first lap will also be saved as the RaceTimer's Best Time, allowing for easy referral to it in TEXT assets.

-LAP2_TIMER (DYNA: game_object__RaceTimer): This timer starts at the beginning of the second lap and is stopped at the conclusion of the second lap. This timer is used to track the lap time (not accounting for milliseconds) of the second lap on the HUD as well as on the results screen. Once a lap is completed and, through a series of conditionals, is determined to be the second lap completed, this timer will be stopped and LAP3_TIMER will be started. The current time on this timer upon completion of the second lap will also be saved as the RaceTimer's Best Time, allowing for easy referral to it in TEXT assets.

-LAP3_TIMER (DYNA: game_object__RaceTimer): This timer starts at the beginning of the third lap and is stopped at the end of the race (or the conclusion of the third lap for courses with more than three laps. This timer is used to track the lap time (not accounting for seconds) of the third lap on the HUD as well as on the results screen. Once a lap is completed and, through a series of conditionals, is determined to be the third lap completed, this timer and TIMERACE_TIMER will be stopped. The current time on this timer upon completion of the third lap will also be saved as the RaceTimer's Best Time, allowing for easy referral to it in TEXT assets.

-LAP1_TIMER_MS (TIMR): This timer starts at the beginning of the race and is reset at the beginning of every lap. At the completion of every lap, the timer is run through a series of COND assets that determine the lap's milliseconds and save them to three different CNTR assets per lap. Each time a CNTR asset has its count set during this millisecond determination progress, a separate CNTR depending on the milliseconds digit gets the same amount added to it. The three different CNTR assets are then used to display the lap milliseconds in the HUD as well as on the results screen. The separate CNTR assets (three, one per millisecond digit) are the sums of the milliseconds digits, used at the end of the race to determine the milliseconds for the final race.

Through the proper utilization of triggers, counters, timers, and conditionals, the lap and timer systems work together to allow for the inclusion of fluent races, accurate timers, and an experience for the racer that feels natural. The final thing regarding the lap and timer systems that the track designer should be educated in is the way in which milliseconds were implemented, as such knowledge is required if the track designer wishes to add more laps. The system must also be explained so that if the track designer wishes to remove milliseconds to allow their level to be played without the use of the AR code that fixes the TimerMillisecondsLeft Conditional, they have the knowledge on how to do so.

The Milliseconds System

As stated before, there are two types of timers used in the CRT's timing system, TIMR assets that count down and track milliseconds, and DYNA assets with the type game_object__RaceTimer that count up and don't track milliseconds. Due to the inability of DYNA RaceTimers to track milliseconds, a TIMR was used to track the milliseconds of the race, that being LAP1_TIMER_MS. Since TIMR assets count down, LAP1_TIMER_MS was set to be a one second timer that restarts upon expiration. This way the timer would infinitely loop and it would be able to be used for all the applications of milliseconds, including the lap times and the race time. The reason it was vital to use one timer for milliseconds is due to the method in which milliseconds of the DYNA RaceTimer were calculated using the milliseconds of the TIMR. The method used to calculate the DYNA RaceTimer's milliseconds using the milliseconds left of LAP1_TIMER_MS is detailed below:

-As LAP1_TIMER_MS is started at the beginning of the race and restarts upon completion of each lap, the timer is always in sync with the lap time, meaning that whenever LAP1_TIMER_MS had a millisecond value of 000, so did the RaceTimer it was tracking milliseconds for. This means that the relation of MS(LAP1_MS_TIMER) + MS(RaceTimer) = 1, and more importantly 1 – MS(LAP1_MS_TIMER) = MS(RaceTimer)

-Upon completion of a lap, LAP1_MS_TIMER is ran through a series of conditionals before determining the milliseconds time and displaying it on the lap time textbox. The conditionals check for the tenths digit, then the hundredths digit, then finally the thousandths digit by checking if the milliseconds left are greater than or equal to 100, 200, 300… 10, 20 30… and 1, 2, 3…, subtracting the amount that the conditional checks for from LAP1_TIMER_MS, setting the count of <u>all</u> of the lap millisecond counters to [10 – the number the conditional checks for, ignoring zeros], and incrementing the appropriate FINAL_MS counter by that same amount
-After LAP_COUNTER has been ran through the LAP_CONDs to determine which lap was just completed, the corresponding counters used for that lap's millisecond tracking are disabled so that they may not be overwritten during the next millisecond calculation. The timer is then reset and ran again at the start of the next lap
-At the end of the race, the FINAL_MS counters are ran through conditionals that do proper addition so that no milliseconds placement for the final time is greater than ten.

The current method used to determine the milliseconds for a lap is at least thirty, with there being ten conditionals per digit of milliseconds tracked. Using the same timer to track the milliseconds for all of the laps and the total race saves ninety COND assets from being used, as each TIMR asset would require thirty conditionals to accurately determine milliseconds using the current method. This is also the reason LAP1_MS_TIMER has "LAP1" in the name; Although it is used for all laps, the timer was initially created and started implementation before optimizations were developed.

With proper knowledge on how the milliseconds system functions, the last modifications to be discussed before going into modification of the CRT are the HUD changes, the custom pause menu, and the results screen.

<u>The HUD, Pause Menu, and Results Screen</u>

Disclaimer: In this context, the HUD includes the health meter, upgrade bar, token counter, and nitro counter, not the lap counter, timer, and lap time textboxes.

The last aspects of the CRT to discuss before going into modifications that a track designer might want to make to their courses are the custom HUD, pause menu, and results screen. All of these work rather simply and modifications to them are not difficult to make, although they may seem complicated upon a first glance if the track designer hasn't worked with custom menus involving the use of DYNA assets.

To start, the removal of the health meter, upgrade bar, and extras counter from the HUD are all done with separate DPAT assets:

-HUD_COMBO_DISABLE_DISP: Disables combo text on HUD
-HUD_EXTRAS_DISABLE_DISP: Disables extras counter on HUD
-HUD_HEALTH_DISABLE_DISP: Disables health meter on HUD
-HUD_UPGRADE_DISABLE_DISP: Disables upgrade bar and current manlies on HUD

These each target an asset in the MN files that are responsible for the presence of each associated asset on the HUD. A similar method is used to disable the upgrade menu and the pause menu, however this is combined into one DPAT asset, that being PAUSE/UPGRADE_DISABLE_DISP.

Speaking of which, it's time to delve into the graceful simplicity of the implementation of the custom pause menu and results screens. Firstly diving into the pause screen, a small group of assets work together to ensure the pause screen looks and functions as expected:

-PAUSE_START_DYNA (DYNA: ui__text): UIFocusOnSelected on ScenePrepare; opens the pause menu. Makes pressing start make the current timer HUD elements invisible, the pause options visible, UIFocusOnSelects PAUSE_MENU_DYNA, and UIFocusOffUnselects self.

-PAUSE_MENU_DYNA (DYNA: ui__text): Serves functions of pause menu. Makes pressing start make the pause options invisible, the timer HUD elements visible, UiFocusOnSelects PAUSE_START_DYNA, and UiFocusOffUnselects self. Makes pressing Z/Select/Back restart the race by ForceSceneResetting RACING_DISP

-PAUSE_OPTIONS_TEXTBOX (DYNA: game_object__text_box): Textbox that the options menu text explaining the functions of the start button and the Z button are.

-PAUSE_TEXTBOX (DYNA: game_object__text_box): Textbox that says PAUSE at the top of the pause screen.

-PAUSE_TEXT (DYNA: hud__text): Text that says "PAUSE"; used as the first argument when PAUSE_START_DYNA makes PAUSE_TEXTBOX visible.

-PAUSE_OPTIONS_TEXT (DYNA: hud__text): Text that says the menu options in the pause menu; used as the first argument when PAUSE_START_DYNA is called to make PAUSE_OPTIONS_TEXTBOX visible.

So at its core, the pause menu is an interaction between two DYNAs, PAUSE_START_DYNA and PAUSE_MENU_DYNA. At the beginning of the race, PAUSE_START_DYNA (START) is enabled and PAUSE_MENU_DYNA (MENU) is disabled. When the racer presses the start button, START disables itself and enabled MENU while making a textbox with menu options visible. Pressing the start button in this state will disable MENU, enable START, and make the textbox with the menu options invisible. Pressing Z will restart the level using the ForceSceneReset event sent to RACING_DISP.

This concept of using DYNA assets to allow for button commands and making textboxes visible for a menu is also used for the results screen. Similar to the pause screen, the results screen is composed of a small amount of assets:

-RESULTS_MENU_DYNA (DYNA: ui__text): UiFocusOnSelected shortly after completion of race; makes pressing the start button reset the level so that the racer may race again

-RESULTS_MENU_TEXT (TEXT): Text that tells the racer that pressing start at the results screen will restart the race

-RESULTS MENU TEXTBOX (DYNA: game_object__text_box): Textbox that is used behind RESULTS_MENU_TEXT

-RESULTS_TEXT/_01/_02/_03 (TEXT): Text used on the results screen. Split into different text assets so that they may be aligned on the results screen in a pleasing manner

-RESULTS_TEXTBOX (DYNA: game_object__text_box): Textbox that is used behind RESULTS_TEXT/_01/_02/_03. Displayed while using RESULTS_TEXT as its argument

-RESULTS_TEXTBOX_01 (DYNA: game_object__text_box): Invisible and used for proper placement; displayed while using RESULTS_TEXT_01 as its argument

-RESULTS_TEXTBOX_02 (DYNA: game_object__text_box): Invisible and used for proper placement; displayed while using RESULTS_TEXT_02 as its argument

-RESULTS_TEXTBOX_03 (DYNA: game_object__text_box): Invisible and used for proper placement; displayed while using RESULTS_TEXT_03 as its argument

-RESULTS_TEXTBOX_GROUP (GRUP): Group containing the RESULTS_TEXTBOX assets

The results screen is much less complicated than the pause menu in terms of its interactions with other assets, as it is all controlled by RESULTS_MENU_DYNA, which only have pressing the start button fade out the screen and ForceSceneReset. Where complications within the results menu may arise are from the numerous TEXT and DYNA: game_object__text_box assets are used to display seemingly two textboxes. The reality of the matter is that this was done so that proper spacing between characters in the lap and race time textbox was to my liking and has nothing to do in terms of the functionality of the results menu. This overall will make changes that require altering the results screen slightly more complicated than if it were one TEXT asset, but as no spacing within the DYNA: game_object__text_box would need to be modified by the track designer, it is a tradeoff that I'm willing to make.

With the knowledge on what the most important assets do and their relationships to other assets, it's time to discuss what would need to be done to make certain modifications that a track designer would want to make to accommodate for their mod or track.

Addition/Removal of Checkpoint Triggers

The addition or removal of checkpoint triggers is perhaps the most common thing a track designer would want to do, allowing them to bring cohesion to their longer tracks and remove clutter from shorter tracks. Not many assets need to be altered to be modified to account for the addition of checkpoint triggers, only the following is needed to be done:

-Addition of checkpoint trigger that has the Links…
    >EnterPlayer => Increment => CP_COUNTER
    >EnterPlayer => Enable => [NEXT CHECKPOINT TRIGGER]
    >EnterPlayer => Disable => [SELF]
-Modification of CP_COND, changing the EvaluationAmount to account for the new number of checkpoint triggers
-Addition of the new checkpoint trigger to CP_TRIGGER_GROUP

The best way to create a checkpoint with the appropriate Links is to copy and paste the most recent checkpoint trigger and edit the links to set what trigger needs to be enabled and to disable itself. Doing this will keep with the checkpoint naming convention of CP_TRIGGER_0X, and as CP_TRIGGER_07 is currently set up to enable CP_TRIGGER_08, this would be the best method for organization and consistency. The addition of the new checkpoint trigger to CP_TRIGGER_GROUP is for the fail-safe that makes passing the finish line without CP_COND returning true. As the fail-safe disables all triggers within CP_TRIGGER_GROUP and enabled CP_TRIGGER_01, it is important to have all checkpoint triggers be in this group to ensure the fail-safe functions properly. For the modification of CP_COND, the track designer can really go whereever with it. They can change the Operation to be GREATER_THAN_OR_EQUAL_TO and set the EvaluationAmount to a number lower than the amount of checkpoint triggers if they wish to allow for shortcuts within the course, however by default CP_COND uses the operation of EQUAL_TO and the EvaluationAmount is set equal to the amount of default checkpoint triggers.

That's all that is required to be done to add more checkpoint triggers. For the removal of checkpoint triggers, it's much simpler:

-Modification of CP_COND, changing the EvaluationAmount to account for the new number of checkpoint triggers

Modifying the CP_TRIGGER_GROUP isn't required as nothing new is being added, and as the amount of triggers is being modified, CP_COND must be modified to account for the new number of checkpoint triggers. The same mentality described in the previous paragraph about what limitations the track designer can choose to set with modification of CP_COND applies here as well, with the track designer able to set CP_COND's Operation and EvaluationAmount to whatever they see fit for the course.

Addition/Removal of Laps

With shorter tracks, it's not uncommon for the track designer to want to add more laps to reach a desired total race time. As discussed in the informative sections about the lap system and the milliseconds system, there is a lot that connects to three laps being the amount in the race, so a good amount of assets must be either added or altered to functionally integrate additional laps. The modifications and additions that will be listed are stated as if the track designer only wishes to add an additional lap, however they may be done multiple times to add as many laps as the track designer desires. With this being said, these are the additions and alterations required to add an additional lap, taking into account milliseconds and the results screen:

-