SmartPoint – Checkpoint and Spawning System for Unity

A Product by SirisGames

Last Edited: January 15th, 2022

SmartPoint is a checkpoint and spawning system designed for 3D development in Unity.

Send me an email at sirisgamesstudios@gmail.com if any of the following apply to you:

- You still have questions about the system after reading this documentation.
- You have encountered a bug.
- There's a feature you'd like to see added or changed.
- You're desperate for conversation.

Software Requirements

SmartPoint was built to be compatible with all pipelines post 2019.4.0f1. This includes, 3D, URP, and HDRP. The materials in the example scenes however use URP shaders and must be manually switched to compatible shaders if opened in 3D or HDRP. This system was built in version 2021.1.23f1 and operates optimally in versions post 2020.3.1f1.

Use Cases

Please read to see if this asset applies to your development needs.

On its own, the checkpoint controller is works optimally as a checkpoint system for a single player. Consider the flag system in Mario, or the bonfire system in Dark Souls. The checkpoints can be activated using the preset activation modes or can be activated through the public methods contained in CheckpointController.cs.

The looping mode works for cases such as a racetrack, or any situation where a circuit might be navigated multiple times. Using the public methods available, it is possible to determine how far the next checkpoint is from the player, or if the player is moving in the wrong direction.

For the spawn system, this opens the system up to objects beyond just the player. With the different spawn modes, this system can be used to refill ammunition, shoot arrows from a wall, spawn enemies at regular intervals while also limiting the number of entities that can spawn, etc... This system can also spawn GameObjects with particle systems, such as explosions or wisps.

If after reading this documentation you're unsure if it's a fit for your use case, send an email to the address listed above and I'll try to get back to you as soon as possible.

CheckPoint

CheckPoints are non-Monobehaviour objects which store data about checkpoints. CheckPoints have 5 attributes which can be accessed via public getters and setters: Index, active state, position, direction, and bounds.

These attributes can be modified in the inspector before runtime or can be modified in script during runtime.

Index: Count starts from 0. Determined by the order in the inspector. Accessed by **GetIndex().**

Active State: Boolean to represent whether the checkpoint has been activated. Accessed by **GetActive()** and **SetActive(bool)**.

Position: Checkpoints position relative to the *CheckpointController*. Accessed by **GetPosition(), GetAbsolutePosition(), SetPosition(Vector3),** and **SetAbsolutePosition(Vector3);**

Direction: Only used for the Align Entity Teleport setting. Accessed by **GetDirection()** and **SetDirection(float).**

Bounds: The bounds of the checkpoint's collider. Used for *OnCollision* events and for *ActivateOnCollision* mode Accessed by **GetBounds()** and **SetBounds(Bounds), SetBoundSize(Vector3)** and **SetBoundCenter(Vector3).**

CheckpointController

Modes

Entity Mode

- Single Entity: Only 1 entity can be configured to interact with the system.
- Multiple Entity: Entities set in the inspector dropdown will be configured to interact with the system.
- Tag Mode: Entities with the selected tag will be configured to interact with the system. This can be a single entity as well, for example, “Player”.

Activation Mode

- Manual Activation: Checkpoints will only activate through scripts configured by the user.
- Always Active: All checkpoints will always be active and cannot be deactivated.
- Activate On Collision: Inactive checkpoints will activate when an entity collides with their trigger box.
- Proximity: Checkpoints will activate if any entity is within *distance* units from their center.

Activation Order

- None: Checkpoints have no ordering constraints.
- Sequential: A checkpoint can only be activated if the directly previous checkpoint is also active. For example, checkpoint 3 can only activate is checkpoint 2 is active. In this example however, the state of checkpoint 1 does not affect checkpoint 3.

Collision Mode

- None: No events will be invoked on collision
- Both: On collision with either active or inactive checkpoints, the OnCollision event will be invoked.
- Active Checkpoints: On collision with active checkpoints, the OnCollision event will be invoked.
- Inactive Checkpoints: On collision with inactive checkpoints, the OnCollision event will be invoked.

Activation Settings

Deactivate On Collision: If true, when a collision occurs with an active checkpoint, that checkpoint will be disabled. The checkpoint will only be disabled under valid conditions. For example, if AlwaysActive is selected, or Singleton mode is enabled, checkpoints cannot be disabled.

Align Entity Teleport: If true, when an entity teleports to a checkpoint, its forward direction (positive x-axis), will be aligned with the checkpoint's *direction*. In other words, it will be rotated about the y-axis.

Create Objects in Scene: If true, create an object for each checkpoint as a child of the checkpoint controller. Setting this to true provides the option to Add Trigger to the object, which creates a trigger collider capable of colliding with any configured entities, and Add Rigidbody, which adds a Rigidbody component with gravity disabled.

Singleton: If true, only one checkpoint can be active at any time.

Looping: Looping can only be set to true if Sequential mode is selected. If true, upon reaching the final checkpoint, the first checkpoint will deactivate. Once the first checkpoint is activated again, all other checkpoints are disabled and the sequence is reset. See SequentialLoopExample scene for an example of this.

Editor Settings

Hide Gizmos: If true, when CheckpointController is not selected in the hierarchy, checkpoint gizmos will be hidden.

Hide Direction Handle: If true, the direction handle will be hidden from the scene view. It can sometimes get in the way of clicks.

Absolute Position: If true, show the absolute positions of checkpoints in the inspector, instead of relative positions.

GUI Scale: The size of the checkpoint spheres in the scene view. If necessary, feel free to modify the range of the slider in CheckpointHandleEditor.cs

Events

There are 4 events that can be invoked. For examples of how to use these events, see the example scenes included in the project.

OnActivate(Int32)

Int32 is the index of the checkpoint that was activated.

Triggers only when a checkpoint switches from inactive to active. See *SetCheckpointState* for more information.

OnDeactivate(Int32)

Int32 is the index of the checkpoint that was deactivated.

Triggers only when a checkpoint switches from active to inactive. See *SetCheckpointState* for more information.

OnTeleport(Int32, GameObject)

Int32 is the index of the checkpoint that was teleported to.

GameObject is the entity teleported.

OnCollision(Int32, GameObject)

Int32 is the index of the checkpoint that collided. *GameObject* is the entity that collided with the checkpoint.

Triggers when an entity configured in the system collides with a checkpoint. What checkpoints can collide is dependent on *CollisionMode*.

Checkpoints must have a trigger collider attached to collide. Additionally, unity collisions only trigger if one of the GameObjects has a Rigidbody. To add a Rigidbody to the checkpoints, see Activation Settings.

Public Methods

Below are detailed summaries of all public methods which are available through the *CheckpointController.cs* monobehaviour script. Feel free to read through the code yourself to better understand or customize functionality.

1. Teleportation Methods

```
public void TeleportToNearest();
```

Teleports the entity to the nearest active checkpoint. This is calculated from the center of the entity to the checkpoint's position in world space.

Use only when a single entity is assigned to the system (in any mode). This could be using *SingleEntity* mode, or having only a single entity assigned in *Tag* mode or *MultipleEntity* mode.

```
public void TeleportToNearest(GameObject entity);
```

Overload which takes in the entity to teleport.

```
public void TeleportToLatest();
```

Teleports the entity to the active checkpoint with the highest index. The index of a checkpoint is determined by its order in the list in the inspector.

Use only when a single entity is assigned to the system (in any mode). This could be using *SingleEntity* mode, or having only a single entity assigned in *Tag* mode or *MultipleEntity* mode.

```
public void TeleportToLatest(GameObject entity);
```

Overload which takes in the entity to teleport.

```
public void TeleportToRecentlyActivated();
```

Teleports the entity to the most recently activated checkpoint, as long as that checkpoint is still active at the time of teleportation.

Use only when a single entity is assigned to the system (in any mode). This could be using *SingleEntity* mode, or having only a single entity assigned in *Tag* mode or *MultipleEntity* mode.

```
public void TeleportToRecentlyActivated(GameObject entity);
```

Overload which takes in the entity to teleport.

```
public void TeleportToRandom();
```

Teleports the entity to a random active checkpoint.

Use only when a single entity is assigned to the system (in any mode). This could be using *SingleEntity* mode, or having only a single entity assigned in *Tag* mode or *MultipleEntity* mode.

```
public void TeleportToRandom(GameObject entity);
```

Overload which takes in the entity to teleport.

```
public void TeleportToCheckpoint(int index);
```

Teleports the entity to the checkpoint at index. The active state of the checkpoint does not matter.

Use only when a single entity is assigned to the system (in any mode). This could be using *SingleEntity* mode, or having only a single entity assigned in *Tag* mode or *MultipleEntity* mode.

```
public void TeleportToCheckpoint(int index, GameObject entity);
```

Overload which takes in the entity to teleport.

2. Getting Checkpoints

```
public CheckPoint GetNearestCheckpoint();
```

Returns the nearest active checkpoint to the assigned entity.

Use only when a single entity is assigned to the system (in any mode). This could be using *SingleEntity* mode, or having only a single entity assigned in *Tag* mode or *MultipleEntity* mode.

```
public CheckPoint GetNearestCheckpoint(Vector3 position);
```

Overload which takes in a position to calculate the nearest checkpoint from. This position could be, for example, the position of multiple entities in a race in order to determine ranking.

```
public CheckPoint GetLatestCheckpoint();
```

Returns the active checkpoint with the highest index. The index of a checkpoint is determined by its order in the list in the inspector.

```
public CheckPoint GetRandomCheckpoint();
```

Returns a random active checkpoint.

```
public CheckPoint GetRecentlyActivatedCheckpoint();
```

Returns the most recently activated checkpoint, as long as that checkpoint is still active.

```
public CheckPoint GetCheckpoint(int index);
```

Returns the checkpoint at index *index* in list, regardless of active state.

```
public CheckPoint[] GetActiveCheckpoints();
```

Returns an array of all active checkpoints.

```
public CheckPoint[] GetInactiveCheckpoints();
```

Returns an array of all inactive checkpoints.

3. Adding and Removing Checkpoints

```
public CheckPoint AddCheckpoint(Vector3 pos, bool activeState, Space space =
Space.World);
public CheckPoint AddCheckpoint(Vector3 pos, bool activeState, Bounds bounds,
Space space = Space.World);
```

Add a checkpoint to the list of checkpoints.

pos is the position of the checkpoint. This defaults to world space but can specified in via the optional parameter *space* using either Space.Self, or Space.World.
activeState is the starting state of the checkpoint. This bypasses OnActivate and OnDeactive events. To trigger the event callbacks on initialization, set the checkpoints state to be the opposite of your desired state, then call SetCheckpointState(int index, bool state) on your checkpoint.
bounds is the bounds for the collision/trigger box of the checkpoint.

Removing checkpoints has yet to be implemented. Send me an email if this is something you'd like to see added, otherwise it likely won't get done. Just something like "Add a function for removing checkpoints please <3". My email's at the top of this manual.
Or you could post a review with this comment on the asset store page, whatever floats your boat.

4. Setting Checkpoints' States

```
public bool SetCheckpointState(int index, bool state);
```

Set checkpoint's active state at *index*, and return true if state was successfully changed. This function will trigger the *OnActivate* or *OnDeactivate* callback events only if the checkpoint's state was successfully changed.
If, for example, you attempt to deactivate a checkpoint in *AlwaysActive* mode, or attempt to activate an already active checkpoint, the event will not be invoked, and will return false.

```
public bool GetCheckpointState(int index);
```

Get the active state of checkpoint at *index*.

```
public void EnableAll();
```

Set all checkpoints' active state to true.

```
public void DisableAll();
```

Set all checkpoints' active state to false.

5. Entities

```
public void AddNewEntity(GameObject entity);
```

Only use in *MultipleEntity* mode. Adds gameobject to the list of entities currently assigned to the checkpoint system.

```
public void RemoveEntity(GameObject entity);
```

Only use in *MultipleEntity* mode. Removes gameobject from the list of entities currently assigned to the checkpoint system.

```
public GameObject[] GetEntities();
```

Returns all entities configured in the checkpoint system.

```
public void ReloadEntities();
```

Only use if an entity has been spawned/destroyed in *Tag* mode. Reloads the list of entities currently using checkpoints. If in *MultipleEntity* mode, the list of entities is reset to its initial state.

SpawnController

Modes

Spawn Mode

- Manual: Entities can only be spawned by calling the public method “Spawn()” from another script.
- Single Burst: A set number of entities will spawn on play.
- Constant: Entities will spawn at regular intervals according to the spawn delay set.
- Refill: In refill mode, only one entity will spawn per active checkpoint. Once an assigned checkpoint’s entity has been destroyed, another will respawn after refill delay seconds. This is useful for situations where resources should be available at specific positions, such as ammunition or power ups.

In both *Constant* and *Refill* modes, the option for an Exit Condition becomes available. If the exit condition is met (<*input*> entities have spawned or <*time*> seconds have passed), the script will stop all coroutines and no longer spawn anymore.

Additionally, the Enable Maximum option becomes available. If true, only <*input*> number of entities can be spawned at one time in the scene.

Spawn Area

- Point: Entities will spawn exactly on this point.
- Circle: Entities will spawn outside the inner (red) radius, but within the outer (green) radius.
- Rect: Entities will spawn within the rectangle’s outlined area.
- Sphere: Entities will spawn within the sphere’s volume.
- Rect Prism: Entities will spawn within the rectangular prism’s volume.
- Use Checkpoint Collider: Entities will spawn within each individual checkpoint’s collider bounds. If the bounds are not visible, ensure that you either have *ActivateOnCollision* enabled, or have Add Trigger selected.

Offset is used to change the spawn position relative to the checkpoint’s position.

Spawn Location

- Highest Index: The active checkpoint with the highest index will always be chosen as the spawn location.
- Most Recently Activated: The most recently activated checkpoint will be chosen. For more information on activating checkpoints, see Activation Modes.
- Nearest to Entity: The nearest active checkpoint to Target Entity.
- Random: A random active checkpoint is chosen.

Spawn Direction

Apart from *Face Entity*, the spawn direction set is limited to the XZ plane. In other words, these directions are solely y-axis rotations.

- Custom: Set a custom spawn direction. 0 is forward (z-axis)
- Face Entity: Spawned entities will face the Target Entity. If Include Y Plane is set to true, the entity will spawn facing the target, with rotations applied in both X and Y axes.
- Align with Checkpoint: Spawned entities will adopt the direction of their checkpoint.
- Random: A random y-axis rotation will be applied on spawn.

Target Entity is an option that applies to *Face Entity* and *Nearest to Entity*.

Editor Settings

Hide Gizmos: If true, handles and gizmos will not be displayed in the scene view.

Other Settings

Check for Collision: If true, entities will only spawn if the spawn area has no other objects it might collide with. This is a bit buggy since the program performs a sphere cast without considering the entity's size. It can also be expensive and delay spawn time slightly,

Equal Spawn Chance: If true, all entities in the entity pool will have an equal* chance of spawning.

*Note: for cases where mathematical fractions cannot be accurately represented with two floating points, small rounding occurs. If there are 3 entities for example, one entity will have a chance of 33.34 while the other two will have a chance of 33.33.

Events

There are 2 events that can be invoked. For examples of how to use these events, see the example scenes included in the project.

OnSpawn(GameObject, Int32)

GameObject is the entity spawned.

Int32 is the index of the checkpoint it spawned at. This will be -1 if no checkpoint was used.

Triggers immediately after an entity has been created.

OnExit()

Triggers when the **Exit Condition** has been met.

Runtime Buttons

Restart Spawner

Calls ResetSpawner(). Resets the spawner, restarting all coroutines and forgetting all previously spawned entities. All editor settings remain the same.

Destroy All Entities

Calls DestroyAllEntities(). Destroy all spawned entities. Specifically, those parented to 'SpawnedEntities' in hierarchy.

Public Methods

Below are detailed summaries of all public methods which are available through the *SpawnController.cs* monobehaviour script. Feel free to read through the code yourself to better understand or customize functionality.

1. Spawn Methods

```
public GameObject Spawn();
```

Spawn entity using SpawnController settings. Returns null if unable to spawn.

```
public GameObject Spawn(CheckPoint spawnCheckpoint, bool applyOffset = true);
```

Spawn entity at specific checkpoint. Returns null if unable to spawn.

```
public GameObject Spawn(Vector3 spawnPosition, bool applyOffset = false);
```

Spawn entity at specific position. Returns null if unable to spawn.

```
public void ResetSpawner();
```

Resets the spawner, restarting all coroutines and forgetting all previously spawned entities. All editor settings remain the same.

```
public void DestroySpawnedEntities();
```

Destroy all spawned entities. Specifically, those parented to 'SpawnedEntities' in hierarchy.

```
public void AddEntity(GameObject entity, float spawnChance);
```

Adds an entity to the pool. May affect other spawn chances.

```
public void RemoveEntity(GameObject entity);
```

Removes an entity to the pool.

2. Getters

```
public GameObject GetEntityFromPool();
```

Return an entity based on spawn chance. Spawn chances to not have to sum to 100.

```
public float[] GetSpawnChances();
```

Returns a list of every entity's spawn chance.

```
public GameObject[] GetEntities(bool asArray);
```

Returns an array of the GameObjects in the Entity Pool

```
public List<GameObject> GetEntities();
```

Returns a list of the GameObjects in the Entity Pool

```
public GameObject GetTargetEntity();
```

Returns the target entity set in the inspector. Used only for specific modes.

```
public Transform GetSpawnerParent();
```

Return the parent object in the hierarchy.

```
public float GetSpawnDelay();
```

Get the delay between spawns (Constant Mode).

```
public float GetRefillDelay();
```

Get the delay before refill (Refill Mode).

```
public float GetSpawnMax();
```

Get the max number of spawned entities that can exist at once in the scene (Constant Mode).

```
public float GetRefillMax();
```

Get the max number of refill entities that can exist at once in the scene (Refill Mode).

3. Setters

```
public void SetTargetEntity(Transform transform);
```

Sets the target entity (used only for specific modes).

```
public void SetSpawnDelay(float delay);
```

For Constant spawn mode. Set the delay before the next entity will spawn.

```
public void SetRefillDelay(float delay);
```

For Refill mode. Set the delay before a checkpoint re-spawns an entity.

```
public void SetSpawnMax(int max);
```

For Constant spawn mode. Set the max number of entities that can exist at one time in the scene.

```
public void SetRefillMax(int max);
```

For Refill mode. Set the max number of entities that can exist at one time in the scene.

OutOfBoundsHelper

The out of bounds helper is a short script which acts as a template for any scripts you, the developer, might write. This will hopefully give you a good starting point. Be sure to look at some of the example scripts in the project's example scenes if you're still unclear on how to use this asset.

First, the associated CheckpointController must be defined.

After the checkpoint controller has been found, in the OnTrigger event, the helper checks if the colliding entity is one of the entities listed in the CheckpointController. If it is, then teleport based on the TeleportMode enum.

The TeleportMode enum simply calls the teleport methods in CheckpointController. I've also added the option in case the player has a CharacterController component.

Issues with SmartPoint/Tips and Tricks

- If you are trying to run the example scenes in HDRP or 3D, you'll have to recreate the materials that are used for those scenes. It'll likely be easier to just open SmartPoint in a URP project rather than go through the trouble, but up to you.
- If the labels on the checkpoints (that show the index) are jittering wildly, this is likely because your camera near clipping plane is too close.
- I've disabled CanEditMultipleObjects since it was not working as intended as was causing problems. Currently working on this.
- Unfortunately, within a single spawn system, it is not possible to have individual parameters for area (such as rect size or sphere radius). This is to save on memory and compute time. If you want each spawn point to have a different shape, then this tool is not for you. If you want rectangular prisms with different bounds however, then it is possible to select Use Checkpoint Collider as the *SpawnArea* and then customize each checkpoint's collider. See MultiEntityExample scene to see how this works.
- You might notice that clicking a checkpoint in the editor dirties the scene causing you to have to save every time, even if no changes were made. To disable this, Open CheckPointHandleEditor.cs. From there, simply comment out all calls to:
    - *RegisterCompleteObjectUndo*
    - *FlushUndoRecordObjects*

  There should be 3 and they should be right next to each other in the list helpers region.

- If you find anything else, let me know and I'll add it to this list or fix it. Email at the top.