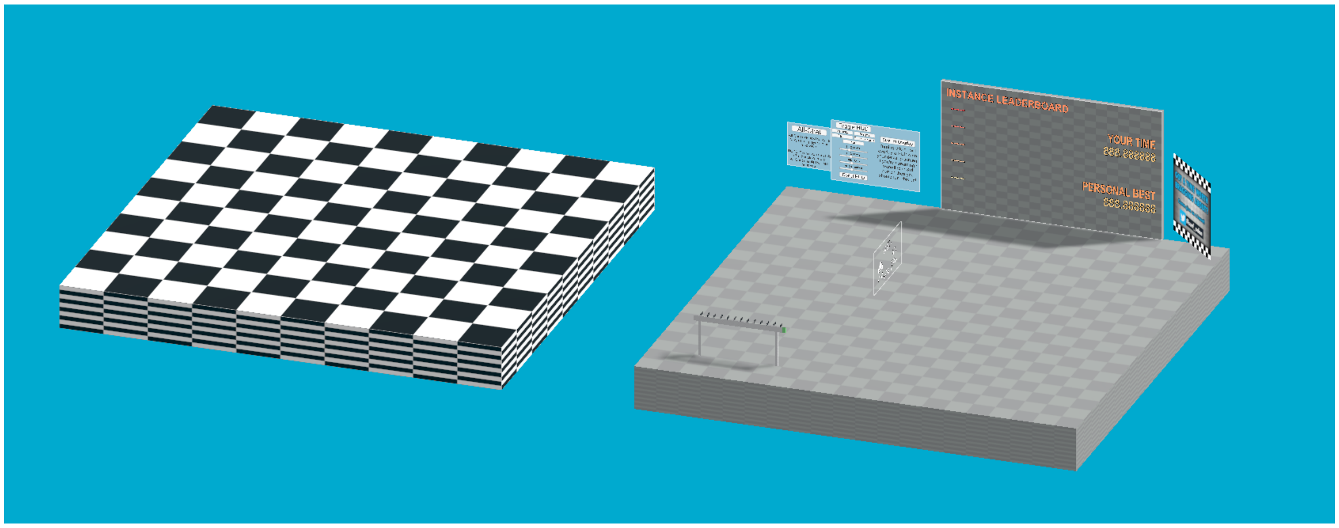
# Surf_Udon Documentation

Created and written by IgbarVonSquid

## Table of Contents



## Quickstart Guide

1. Download the latest SDK3 – Worlds Unitypackage from VRChat (SDK2 will **not** work!)
2. Import the SDK into your Unity project.
3. Find the *VRCWorld* prefab under *VRChat Examples > Prefabs* and drag it into your scene.
4. Bring up the VRChat SDK Control Panel, login, then click the prompts to build the Layers and Collision Matrix.
5. Find the *Surf_Systems* prefab in the *Surf_Udon* folder and drag it into your scene.
6. At this point you should Build & Test the scene to make sure everything works. Simply jump from the starting platform to the finish platform while watching your velocity, time, etc to make sure all the systems work as expected.
7. You can now start building out a basic course. You'll find ramps and boosters in their respective folders under the *Surf_Udon > prefabs* folder. If you ever get lost or confused you can refer to the *Tutorial Scene* in the *Surf_Udon* folder to see how things are setup there.

# Known Issues

**The physics don't work the same as in Source!**

Surf_Udon is still using VRChat's player physics with some added logic on top. Making a whole new system from scratch takes a lot of knowledge and time that I don't currently have. In this version of surf players build speed from boosters instead.

**Players will randomly get bumped off of ramps**

This is one of the issues related to using VRChat's player physics and has to do with slamming into a surface with a lot of velocity which makes VRChat want you bounce you back. The *SurfController* has a built in fix for this when it comes to straight ramps called ramp-alignment: When a player lands on a straight ramp their velocity is lined up with the direction of the ramp so that their perpendicular velocity is set to 0. This prevents the issue most of the time but doesn't prevent it entirely. It also doesn't work for curved and mega ramps. This is why I recommend players not hold strafe or forward on ramps as this can mess with their velocity and make the game bump them off. Boosters can also bump players off of ramps at random because of how they affect player velocity. In these cases the BoostOmni program should be used but it isn't particularly consistent either. I hope to have a better solution to this eventually.

**Curved ramps randomly slow down players**

Curved ramps are the worst of the issues related to the player controller. Sadly nothing can really be done in this case as the ramp-alignment trick won't work here. I recommend that you use curved ramps sparingly and right after checkpoints, as well as using a clamped booster after it so that if players get slowed down they'll be brought back up to speed. Giving players a lot of speed into the ramp also seems to help because going too slow will definitely knock them off.

**Turning your head and strafing decreases speed / Surf seems to work best by not rotating at all**

This another one of the downsides with relying on VRChat's player physics. By rotating your head/body and trying to move in a direction you end up losing speed in the process. This is one of the issues that can actually be fixed but may take some time to come up with a good solution. Please bear with it for now.

**Strafing is too hard on low framerates**

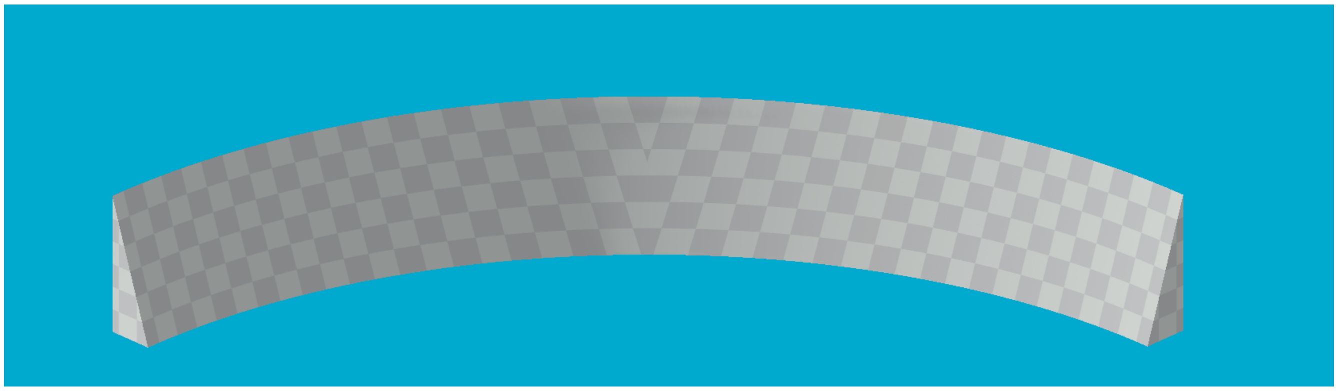
Another issue with VRChat's player physics, see this Canny. Nothing can be done on our end unless we use our own player physics.

**Your public variables are out of order!**

Public variables are ordered by time of creation because you can't reorder them with Udon graphs yet. See this Canny.

**I can't test my map with CyanEmu**

CyanEmu is not designed to replicate VRChat's player physics so surfing won't work in editor. You'll just have to do lots of Build & Tests so I recommend you keep your world very simplistic until the course itself is finished.

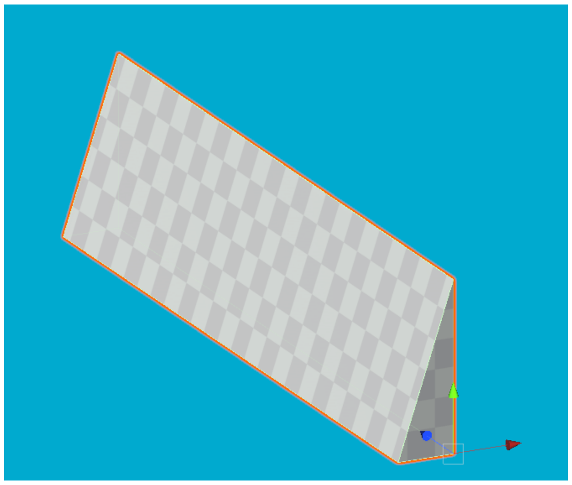


## Ramps Guide

Ramps are what players surf on to complete your course. When players land on a ramp they won't lose momentum like if they had landed on a regular surface. You can find them under *Surf_Udon > prefabs > ramps*. Ramps are defined by two factors:

- On the VRChat side of things, a ramp is any surface angled between 60 to 90 degrees.
- On the Surf_Udon side of things, ramps are recognized by their layer. By default these layers are *11: Environment* for straight ramps and *8: Interactive* for anything else. You can change these layers in the *SurfController* UdonBehaviour.

**Straight Ramps** are treated special because we can use the ramp-alignment fix on them to help prevent players from randomly getting bumped off (See the section on *"Players will randomly get bumped off of ramps"* on page 3 for more details). If you use your own ramp meshes just make sure they're orientated like this with the x axis (red arrow) perpendicular to your ramp:



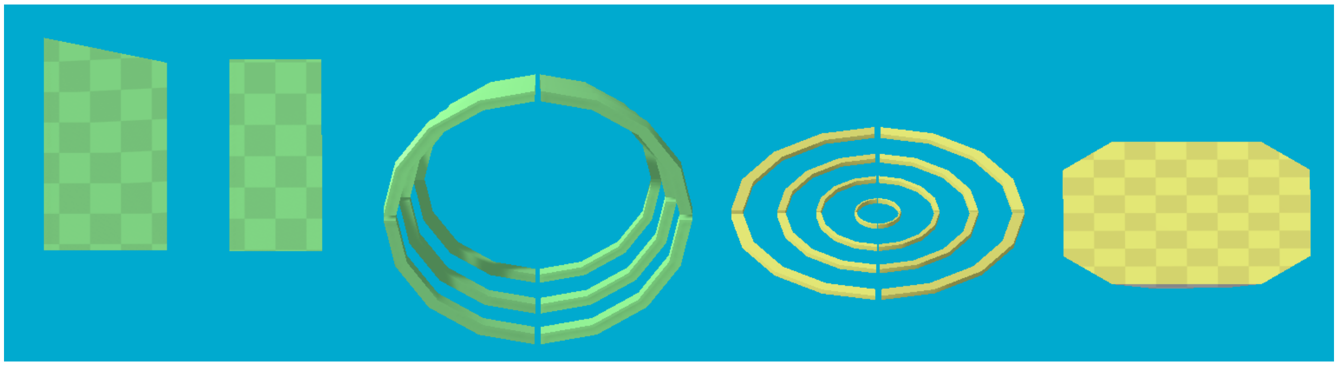
**Curved Ramps** on the other hand can't use ramp-alignment because they're not straight. For this reason players are more likely to get bumped off, especially if going too slow. I recommend using curved ramps sparingly and/or near the beginning of a stage. You should also make sure players are going fairly fast through them and for best results follow it up with a clamped booster.

**Mega Ramps** are made especially tall so that players can surf up them. They should be preceded by a booster to give players the positive Y velocity needed to go up them. The sooner a player hits this ramp the more velocity they'll have. Because of this you need keep in mind that due to differences in framerates and reaction times players will have varying levels of difficulty in getting enough speed to make it up. Make sure to give players plenty of wiggle room for where they have to land.

The last type of surfable object is the **WaterArea** found under *Surf_Udon > prefabs > props*. It's technically a ramp but it allows players to move freely while on it. Note that the collider is a trigger and therefore isn't actually solid to players but instead just disables their gravity while they're on it.

**Other considerations:**

- Don't attach ramps together as this will cause issues for the *SurfController*. You can use the *RampGapFiller* to visually attach two ramps together instead.
- You can easily mirror a ramp by giving it a -1 scale on the X axis.
- Ramps should only be rotated on the Y axis. Rotating on X or Z could lead to unexpected results.
- Don't rely too much on curved ramps because they're prone to slowing down players and/or bumping them off.
- Mega ramps can be difficult for players to use. Make sure to give them a wide margin of error or design the course so that players instinctively hit it as fast as possible.



## Boosters Guide

Boosters are what players use to build up speed on your course. You can find them under *Surf_Udon > prefabs > boosters*. They come in a variety of forms to suit different situations and should be mixed and matched to diversify your course. If you aren't familiar with velocity vector3s then it's important to understand that velocity is handled in global space, not local. This means that the player's rotation doesn't matter; Z positive will always be in the same direction. Also note that for the most part you can swap out the Udon programs to whichever one you need to suit the situation regardless of which booster model it is. For example you're fine to use the *BoostClamp* program on the *BoostLauncher* model.

### Ramp Boosters

**BoostPad** uses the *BoostOmni* program and is designed to go on the side of a ramp and give players more velocity. However, this program has specific behavior designed to help with ramp-alignment and may behave unexpectedly: Setting the X or Z axis to 0 will result in that velocity axis being to 0 instead of adding 0. Refer to the documentation on page 15 for more details.

**BoostLauncher** uses the *BoostLauncher* program and is designed to go at the end of ramps. The *Velocity* variable is added on top of the player's current velocity.

**BoostClamp** uses the *BoostClamp* program and is designed to keep the player's velocity within a set range. This behavior is useful to make sure players are up to speed after difficult sections.

**BoostOverride** uses the *BoostOverride* program and is designed to set the player's velocity directly without doing any math. I don't recommend actually using this but it exists in case you need it.

**Non-Ramp Boosters**

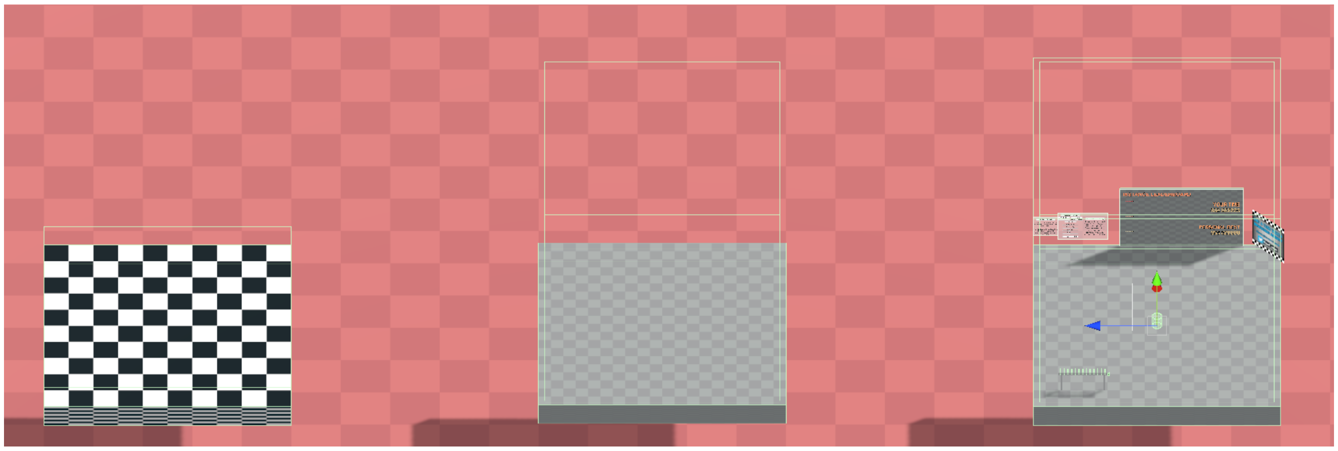
**<u>BoosterRing</u>** uses the *BoostForward* program and is designed to boost players in midair in the direction the booster is facing.

**<u>BoostFloor</u>** uses the *BoostUpward* program and is designed to launch players upward regardless of their current Y velocity.

**<u>BhopPad</u>** also uses the *BoostUpward* program and is used to bounce the player upward based on the Y velocity they hit it with.

**Other considerations:**

- It's very important to use BoostClamps at regular points in your course to make sure players are up to speed. If you don't then players may be frustrated that minor losses of speed lead to them failing the course.
- The BoostLauncher is the easiest to use booster if you just want to add more velocity to the player.
- Boosters will sometimes throw players off of ramps so be careful about how many you're using in order to decrease how much it happens.



## Staging & Teleporting Guide

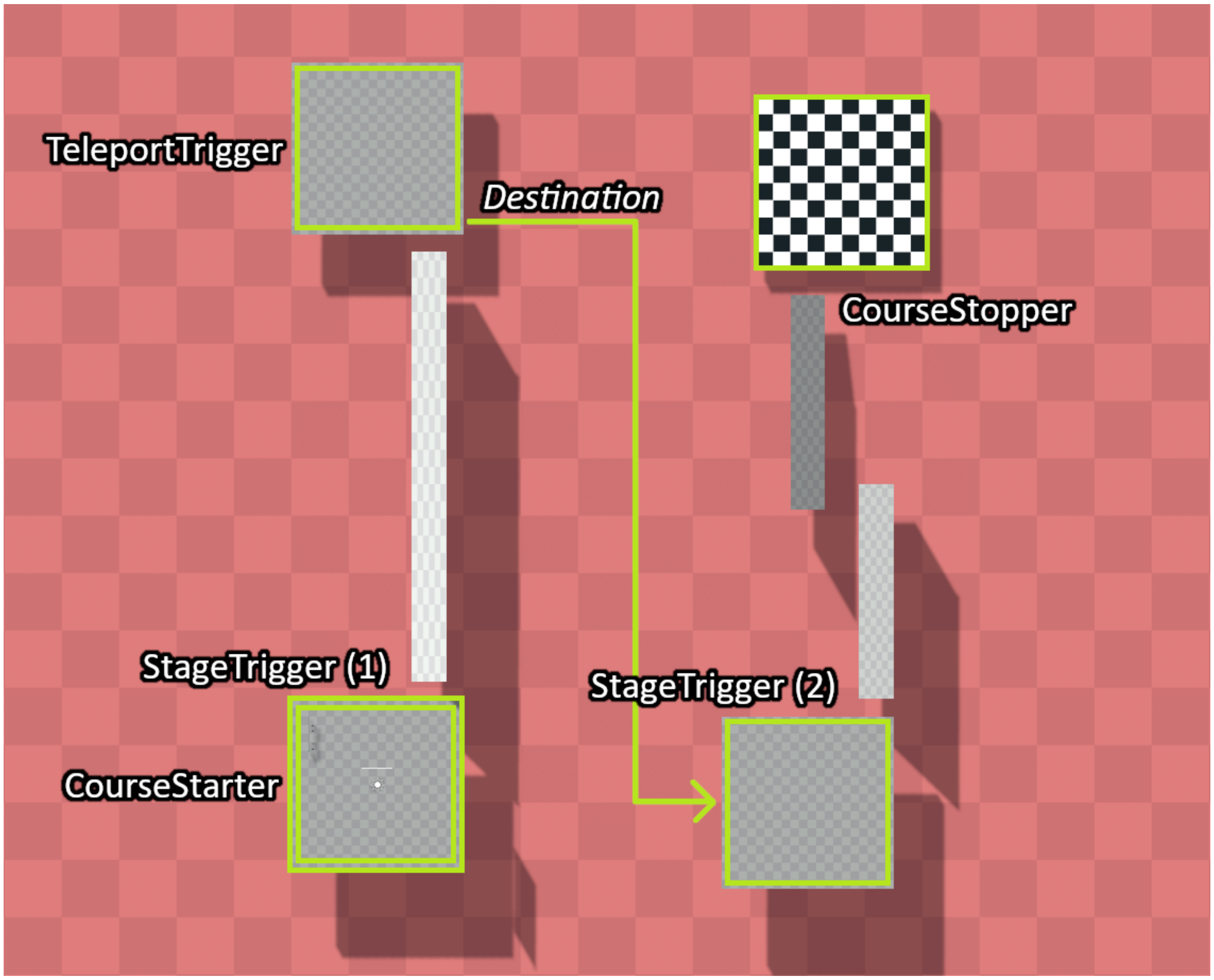
While surf maps can be single stage, you'll probably want to go with a multi-stage layout instead. The three things you'll need are:

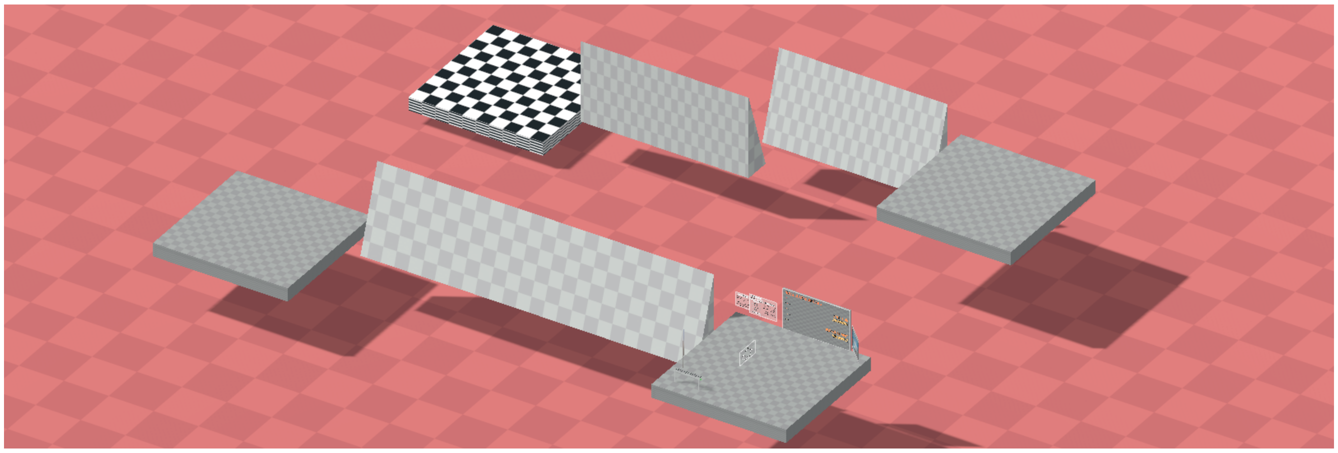
-The **StageTrigger** prefab included in the *Surf_Systems* prefab where you can duplicate it from easily. It can also be found under *prefabs > systems*. Entering into this trigger updates what stage the player is on and moves their respawn point. *RespawnTransform* is what transform to move when entering the stage and should be the *RespawnTransform* included in the *Surf_Systems* prefab.
**Note that the respawn point and the world spawn point should be two different transforms!** Do not use the same for both or else players won't be able to return to the beginning of the course. The *RespawnTransform* is used only for being respawned by obstacles on the course. Make sure to change the *StageNumber* as well.

-The **TeleportTrigger** prefab found under *prefabs > systems*. You can place this at the end of a stage to teleport players to the next stage when they enter the trigger collider. *Destination* is what transform you want to teleport the player to, which in this case should be the transform of the next *StageTrigger*. You can also enable *KeepVelocity* so players will maintain some of their speed through the teleport.

-The **Respawner** prefab found under *prefabs > props*. If the *SurfController* colliders with this the player will be teleported back to the beginning of the stage. If players just fall out of the map they'll be brought back to the beginning of the course so make sure to cover the bottom of your course with these instead. Note that it's on layer *14: PickupNoEnvironment*. If you change the layer of the *SurfController* to something that isn't *13: Pickup* then you should change the layer of this too and update the *RespawnLayer* variable on the *SurfController* to reflect this. Also note that the *SurfController* is by the player's feet and doesn't encompass the entire player collider. This means that respawners **must** hit the bottom of the player in order to trigger the respawn. Overhead respawners will not work!

Here's a visual example of how to setup your course:





## Finishing Touches Checklist

- **Play test, play test, play test!!** Use every booster and ramp, bump into every wall, etc.
- You can use Unity's *Physics Debugger* window to find missing colliders and triggers. You can also use this to easily check if your *TeleportTriggers* and *StageTriggers* properly fit onto your platforms: They should be slightly smaller than their platform so that bumping into the side of it doesn't count. Players must land on top in order for them to have cleared the stage.
- If a particular section seems somewhat hard for you, it'll be ***much*** harder for everybody else. Consider making it more forgiving so your course is more accessible to more people.
- Play test in VR. Surfing can be harder for VR users because minor head rotations can cause them to slowdown. If slowing down on certain parts is a reoccurring issue then consider using a clamped booster after it to help get players back up to speed.
- Also make sure to test your world at a lower framerate. You can do this by first uploading your world, then: Go to steam, find VRChat, right click it, click *Properties…*, and then in the *Launch Options* put *--fps=30* . This will force the game to run at 30 frames per second which is the lowest players are willing to put up with in most cases. If players are experiencing a framerate lower than this, even in a private instance, then you should spend some extra time making sure the world is optimized. If your world is too hard at low frames try to make it slightly easier for those in public instances and who have bad computers.

- Make sure your *StageController* has the correct amount of stages and your *StageTriggers* have the correct *StageNumber*.
- Make sure your *StreamOverlay* has the name of your world written for the *NameOfWorld* value! This is a good way to get exposure from streamers who have the overlay enabled.
- If your world name uses an underscore then make sure you have the name of your world *without* the underscore in your world description. This makes it easier for people to find your world from the in-game search.
- Give your world the “Surf” tag so people can find it from that as well.

# Reference Documentation for All Prefabs

## Booster Prefabs

**BhopPad** – This booster uses the *BoostUpward* program with *UpwardVelocity* set to 1 and *InvertVelocity* enabled by default. You can use it to bounce the player upward with no loss to lateral velocity. The bounce will only happen if players land on top; if they hit the side then no bounce will occur. Note that this object can't be stood on top of so if you want a solid platform with bounce you should put the *BouncyPhysMaterial* on a solid platform instead.

- **UpwardVelocity** – What the player's Y velocity will be set to when they land on top. This ignores the player's current Y velocity entirely and makes no changes to X and Z velocity. If *InvertVelocity* is enabled then this will instead be used to lerp between 0 and the absolute value of the player's current Y velocity. See below-

- **InvertVelocity –** If enabled then the player's current Y velocity when they land on top will be factored into the bounce using *UpwardVelocity* to control the strength. The equation works like this: Take the player's current Y velocity, get it as an absolute value, then lerp between 0 and that value using *UpwardVelocity*. For example if the player hits the top with -24 Y velocity and *UpwardVelocity* is 0.5 then the resulting Y velocity will be 12. If *UpwardVelocity* is 1 then it would be 24. Note that this lerp is unclamped so if you set *UpwardVelocity* higher than 1 the player will actually gain Y velocity with every bounce.

**BoostClamp –** This booster uses the *BoostClamp* program with *MinVelocity* X Y Z set to -Infinity and *MaxVelocity* X Y Z set to Infinity by default. As the name suggests it will clamp the player's velocity based on what values you set.

- **MinVelocity –** Used to clamp the player's minimum velocity with control over each axis. Any axis you don't care about should be set to -Infinity. When working with negative numbers you should set the miniumum velocity using *MaxVelocity* instead. See below-

- **MaxVelocity –** Used to clamp the player's maximum velocity with control over each axis. Any axis you don't care about should be set to Infinity. Note that 'maximum' in this case doesn't necessarily mean what we think it does when working with negative numbers: If the player is moving forward with a velocity of Z -30 and we set the *MaxVelocity* Z to -100 then the player's velocity will turn into -100. This is because -30 is a *higher* number than -100 even though it means we end up gaining speed instead. So if you want to set the player's minimum velocity with a negative number you have to set it as a maximum.

**BoosterRing –** This booster uses the *BoostForward* program with *ForwardVelocity* set to 10, *UpwardVelocity* set to 2, and *OverrideYvelocity* enabled by default. Unlike the other boosters this one adds to your velocity based on the local forward direction of the object itself. This makes it much easier to work with as you don't need to set each velocity axis directly and can instead just rotate the object to control direction.

- **ForwardVelocity –** How much velocity to add to the player's current lateral velocity based on the rotation of this object.

- **UpwardVelocity –** How much velocity to add to the player's current Y velocity. If *OverrideYvelocity* is enabled then this value will instead be directly set as the player's new Y velocity without consideration of the player's current Y velocity. See below-

- **OverrideYvelocity –** If enabled then the player's Y velocity will be overridden instead of added to using *UpwardVelocity* as the new value. You should use this in cases where the player's Y velocity might be near or below 0.

**BoostFloor –** This booster uses the *BoostUpward* program with *UpwardVelocity* set to 10 and *InvertVelocity* disabled by default. You can use it to bounce the player upward with no change to lateral velocity. Refer to the documentation for the *BhopPad* on page 13 for a breakdown of what the variables do.

**BoostLauncher –** This booster uses the *BoostLauncher* program with *Velocity* Y set to 5 and X & Z set to 0 by default. This is designed to add velocity to the player at the end of a ramp without any ramp-alignment considerations.

- **Velocity** – This will be added directly to the player's current velocity.

**BoostOverride** – This booster uses the *BoostOverride* program with *NewVelocity* X Y Z set to 0 by default. You can use this booster to directly set the player's velocity without any consideration of the player's current velocity. As in, the player's velocity is being overridden entirely. In most cases it would likely be better to use *BoostClamp* instead but I've included this program just in-case you need it.

- **NewVelocity –** This will be the player's new velocity. There is no math involved here.

**BoostPad** – This booster uses the *BoosterOmni* program with *Velocity* X Y Z set to 0 by default. This is designed to go on the sides of ramps and boost the player will full control over each axis as well as assist with ramp-alignment. Due to this it may behave unexpectedly in some circumstances, see below-

- **Velocity –** Used to decide how much velocity to add to the player and in what direction. This has a feature to help with ramp-alignment built in for the X and Z axis: If either of these are 0 then instead of adding we set that axis to 0 directly. If both or neither are 0 then the velocity is simply added instead. Note that in all cases the Y velocity is always additive. If you run into a problem where the player suddenly stops when they hit this booster change the 0 to something like 0.01 so it's not exactly 0. If you only want to add velocity without consideration of ramp-alignment then use the *BoostLauncher* program instead.

## Debug Prefabs

**DebugObjectMover –** You can use this prefab to move an object around in-game without having to Build & Test over and over. It uses the numpad for the controls: 8 is forward, 5 is backward, 4 is left, and is 6 is right. You can also move it up and down using 7 for up and 9 for down. The object is moved in 1 meter increments and the new local position will be written into the debug log (You can open the log by pressing Right Shift + ~ + 3). From there you can copy down the position and fix it in Unity. If you're not seeing anything in the log then confirm that your *DisplayName* is correct and that your numlock is enabled. Also note that if the object is set to Batching Static then it won't move.

- **TargetTransform –** The transform of the object you want to move.

- **DisplayName –** Write your VRChat display-name here. This is a fail-safe so that if you accidentally upload your world with the debug still enabled only you will be able to use it.

**DebugPosition –** You can use this prefab to help you figure out where to place objects on the course. When you press the debug key it will print your current position and velocity to the debug log. Uses the *e* key by default.

- **DebugKey –** Which key when pressed will print the information to the log.

**Test Ramp – 60 & 89 degrees –** These are test ramps designed to make sure the surf system still works even at the most extreme angles. Surfing only works on slopes higher than 60 degrees but less than 90 degrees. If you're making any changes to the surf mechanics at all then you should double check that it still works on these ramps.

## Prop Prefabs

**Arrow –** This is purely visual. You can use this to point players in the direction you want them to go. You should use it in places where it's not obvious where players need to land such as after strafing sections where they can't see the end.

**KnifePickup** – A knife. It *cannot* be used to hurt players, it just does cool spinny tricks when you use it. The animations change based on whether the person holding it is on desktop or VR. If you want to replace the knife with your own custom one just name it *KnifeVisual* and make it a child of the *KnifePickup*. Alternatively you can disable the mesh rendering on *KnifeVisual* and just make it a child object of that instead. Most of the logic is on the *KnifeLogic* because ObjectSynced UdonBehaviours can only be set to Continuous. Moving that logic to a Manual sync UdonBehaviour saves on networking.

- **KnifeLogic** – The UdonBehaviour that handles the knife logic. This should be the *KnifeLogic* child object.
- *Custom Event:* **_Respawn –** When this event is called it will reset the knife back to its default position. The *ResetButton* on the *KnifeTable* prefab calls this event.

**KnifeTable** – A table used to hold a collection of knives. Everything is purely visual except for the *ResetButton* which houses the resetting logic-

- **Knives –** An array for all the knives you want to reset when this button is pressed. When you press it you'll take ownership of every knife that isn't currently being held and then call the *_Respawn* event on those knives.

**Leaderboard** – A wall to display the best times of the current instance as well as the player's personal times. It's updated whenever the *YourTime* private variable is changed, usually from the *TimerController* after the player completes the course.

- **LeaderboardTexts** – An array of UiTexts to be used for the leaderboard. Note that it's in reverse order such that index 0 is 5th place and 4 is 1st place.

- **NoTimeMessage** – What to display by default when there's no time available.

- **ScorePbText** – The UiText to display the player's personal best time.

- **ScoreTimeText** – The UiText to display the player's previous time.

**Respawner** – This prefab is an example of a respawner. When the player enters the trigger they'll be brought back to their last *StageTrigger*. Note that the respawning logic is handled by the *SurfController* and not this object. Respawners are designated by their layer (layer *14: PickupNoEnviroment* by default) and must have their colliders set to *Is Trigger*.

**WaterArea** – This prefab is a surfable object that isn't actually a ramp. It uses layer *8: Interactive* by default so that ramp-alignment isn't used on it. The collider is set to *Is Trigger* so that players don't lose their momentum by standing on it. Instead it enters the trigger of the *SurfController* so that it can disable the player's gravity.

## Ramp Prefabs

**RampCurved Long, Medium, & Short –** Curved ramps for redirecting the player's direction. Note that they use layer *8: Interactive* by default. Use these sparingly as they are known to have issues, see the section on "*Curved ramps randomly slow down players*" on page 3 for more details.

**RampCurvedMega –** A curved version of the *RampMega* used for surfing upward.

**RampGapFiller –** A very small ramp used to bridge the gap between ramps. Because ramps can't directly touch each other you can put this in between to fill in the gap. Note that because its on the *Default* layer it isn't treated like a ramp by the *SurfController*.

**RampMega –** A ramp that's tall enough to be climbed up given the player has a positive Y velocity. Note that this uses the same layer as the curved ramps (layer *8: Interactive* by default) because we don't want to apply the ramp-alignment fix here.

**RampStraight Long, Medium, & Short –** Straight ramps. Note that they use layer *11: Environment* by default so that the *SurfController* will use ramp-alignment on it. See the section on "*Players will randomly get bumped off of ramps*" on page 3 for more details.

## System Prefabs

**CourseStarter –** A trigger volume used to start the course timer. When the player exits the trigger the timer is started. If they re-enter then the timer is stopped. You should make sure the trigger is properly sized around your spawn platform.

- **TimerController –** The *TimerController* UdonBehaviour. The *CourseStarter* fires the *_Start* and *_Reset* Custom Events on this UdonBehaviour.

**CourseStopper –** A trigger volume used to stop the course timer and then teleport the player. When the player enters the trigger the timer is stopped. You should make sure the trigger is properly sized around your final platform.

- **TimerController –** The *TimerController* UdonBehaviour. The *CourseStopper* fires the *_Stop* Custom Event on this UdonBehaviour.
- **TeleportPlayer –** If enabled then the player will be teleported to the *Destination* transform upon entering the trigger. Enabled by default.
- **Destination –** The transform the player will be teleported to when they enter the *CourseStopper* and *TeleportPlayer* is enabled.

**DisplayInput –** A system to display your input on the HUD and stream overlay.

- **InputAnimatorHUD** – The *InputDisplay* animator on the *HUD* prefab.
- **InputAnimatorOverlay** – The *InputDisplay* animator on the *StreamOverlay* prefab.

**DisplayVelocity** – A system to display your current velocity on the HUD and stream overlay.

- **VelocityText** – The UiText on the *HUD prefab* to display your velocity on.
- **DecimalRounding** – Controls the rounding for the velocity. Default is 1000 (will only show 3 digits past the decimal point). Setting to 1 will round to whole numbers.
- **UpdateIntervalInSec** – How often to update the velocity in seconds. Default is 0.1. Setting to 0 will update the velocity every frame and 1 will update once every second.
- **VelocityTextOverlay** – The UiText on the *StreamOverlay* prefab to display your velocity on.

**HUD** – A system to display information in your field of vision by moving a canvas in front of you every frame. The attached animator is what controls the HUD elements, which in turn is controlled by the *HUDSettings* in the *Menus* prefab-

**Menus** – The settings menu used to control HUD options, stream overlay, and player voice system.

- **HudSettings** – Controls toggling HUD elements through the HUD's animator.
- **VoiceRangeSettings** – Controls the player voice settings. *MaxVoiceRange* controls how far you'll hear other players. The default is 1000000 which is the highest VRChat will let you set it. Disable *OnByDefault* if you don't want the maximum voice range enabled by default.

**StageController** – A system that handles displaying your current stage on the HUD and stream overlay.

- **StageText** – The UiText on the *HUD* prefab that displays your current stage.
- **StageTextOverlay** – The UiText on the *StreamOverlay* prefab that displays your current stage.
- **NumberOfStages** – The maximum number of stages on your course. Make sure you change this to properly reflect how many stages your course has.
- **StageMaxText** – The UiText on the *HUD* prefab that displays the max number of stages.
- **StageMaxTextOverlay** – The UiText on the *StreamOverlay* prefab that displays the max number of stages.

**StageTrigger** – A trigger volume used to update your stage on the *StageController*, move your respawn point, and move the *Menus*. It comes with a (1) in the name by default so you can easily make duplicates with matching numbers in the name.

- **RespawnTransform** – The player's respawn transform which we move to this stage. This should be the *RespawnTransform* included in the *Surf_Systems* prefab.

- **StageController** – The UdonBehaviour for the *StageController*.

- **StageNumber** – What stage the player is on when they enter the trigger, which is then updated to the *StageController*. Make sure you change this to accurately reflect which stage this is.

- **MoveMenus** – If enabled then the *Menus* will be moved to this stage, enabled by default.

- **MenusTransform** – The transform for the *Menus* that we move if *MoveMenus* is enabled.

- **MenusNewPosition** – The transform we're moving the *Menus* to. By default it's the child object of this *StageTrigger*.

**StreamOverlay** – A system to display information on your desktop which can then be picked up by the stream camera. It should be disabled by default and enabled by players who want it from the *Menus*.

- **WorldNameText –** The UiText used to display the name of your world on the *StreamOverlay*.

- **NameOfWorld** – The name of your world. Make sure you update this to accurately show the name of your world on stream cameras.

**SurfController** – The system that handles the surf physics. Note that this is on layer *13: Pickup* and that changing it off of this layer may mean you have to change some of the layer variables as well.

- **RampStraightLayer** – The layer used to detect straight ramps that should use the ramp-alignment feature (See the section on "*Players will randomly get bumped off of ramps"* on page 3 for more details). Default is 11 (*Environment*).

- **AirStrafeSpeedMin** – The minimum strafe speed players will have while in the air. Default is 10.

- **GravityStrength** – How strong gravity is for the player while in the air. Default is 1. You can change this variable at any time and it will update for the player automatically.

- **RespawnLayer** – The layer used for objects that should instantly respawn the player to their last stage checkpoint. Default is 14 (*PickupNoEnvironment)*. Note that if you change the *SurfController*'s layer off of *Pickup* then you should change this layer as well because *PickupNoEnvironment* will only collide with pickups.

- **RespawnTransform** – The player's respawn transform separate from their world spawn. We move the player back to this transform whenever they get respawned from coming in contact with a respawner.

- **MoveSpeedGround** – The player's walk, run, and strafe speed when they're on the ground. Default is 5.

- **MoveSpeedAir** – The player's walk and run speed while in the air (strafe speed is handled separately). Default is 10.

- **KeepVelocityOnRespawn** – If enabled the player will keep some of their velocity if they get respawned. Y velocity will converted to an absolute value and halved to prevent infinite height gain. How much of the player's velocity is kept is dictated by the *RespawnVelocityFactor* variable. Disabled by default.

- **StrafeScaleFactor** – Controls how much of the player's lateral velocity should be converted into strafe speed. For example if a player is going 100 m/s forward and *StrafeScaleFactor* is 0.5, then the player's strafe speed is set to 50 m/s. Default is 0.5. Increasing this too high will lead to player's gaining infinite speed from strafing.

- **AirStrafeSpeedMax –** Limits how high the player's strafe speed can get. Default is Infinity. You should make this an actual number if you set the *StrafeScaleFactor* too high to prevent players from gaining infinite speed.

- **RampOtherLayer** – The layer used to detect ramps that shouldn't use the ramp-alignment feature. Default is 8 (*Interactive*).

- **RespawnVelocityFactor** – How much of the player's velocity to maintain after they get respawned. Default is 0.65. This value is lerped between 0 velocity and the player's previous velocity before they got respawned. A value of 1 would maintain all of the player's velocity. Note that this lerp is unclamped so setting it higher than 1 will lead to player's gaining additional velocity.

- *Custom Event:* **_Disable** – Call this event to safely disable the surf system. It will set the player's walk, run, and strafe speed to *MoveGroundSpeed* and set the player's gravity to *GravityStrength*, then turn the gameobject off. To turn surf back on again just enable the gameobject.

**TeleportTrigger** – A system to handle teleporting players with options to keep velocity.

- **Destination** – The transform to teleport the player to. I recommended teleporting the player directly to the transform of a *StageTrigger*.

- **KeepVelocity** – If enabled the player will keep their velocity through the teleport. Disabled by default.

- **yVelocityScale** – How much of the player's Y velocity should be carried through the teleport if *KeepVelocity* is enabled. This value is used to lerp between the absolute value of the player's current Y velocity and the *yVelocityMin*. A value of 0 will use the *yVelocityMin*, a value of 1 will use the absolute value of the player's Y velocity, and 0.5 will be halfway in-between.

- **YvelocityMin** – The minimum amount of Y velocity the player should have after teleporting if *KeepVelocity* is enabled. Default is 0.

**TimerController** – The system that handles keeping track of the player's time on the course and displaying it.

- **LiveTimeText** – The UiText on the *HUD* prefab to display your time on.

- **NoTimeMessage** – What string to display when there's no time available.

- **UpdateIntervalInSec** – How often to update the timer, in seconds. default is 0.05. Setting to 0 will update the time every frame, 1 will update once every second. Note that the time is still saved every frame internally regardless of this setting.

- **DecimalRounding** – Controls the rounding for the time displays. default is 1000 (will only show 3 digits past the decimal point). Setting to 1 will round to whole numbers.

- **LastTimeText** – The UiText on the *HUD* prefab to display your previous time on.

- **PbTimeText** – The UiText on the *HUD* prefab to display your personal best time on.
- **LiveTimeTextOverlay** – The UiText on the *StreamOverlay* prefab to display your time on.
- **LastTimeTextOverlay** – The UiText on the *StreamOverlay* prefab to display your previous time on.
- **PbTimeTextOverlay** – The UiText on the *StreamOverlay* prefab to display your personal best time on.
- **Leaderboard** – The *Leaderboard* UdonBehaviour.
- *Custom Event:* **_Start** – Starts the timer.
- *Custom Event:* **_Reset** – Stops the timer and puts it back to 0.
- *Custom Event:* **_Restart** – Puts the timer back to 0 but doesn't stop it. Not currently used but included if you have a use-case for it.
- *Custom Event:* **_Stop** – Stops the timer and saves your time, then passes that information to the *Leaderboard*.