Soldat Mapping Tutorial

How to make a decent map

Authors: Viral

Last update: August 24, 2022

Contents

1 Introduction

This document gathers all useful tips, tricks and general mapping rules that the authors of this paper have discovered throughout their beautiful Soldat career. It's supposed to help mappers create high quality maps with minimal effort. This tutorial covers:

1. The use of Soldat Polyworks.
2. The general concept of map creation.
3. The range of useful tips and solutions to common problems.

Please note that statements presented here are subjective and based entirely on authors' experience. This means "no way" should be read as "as far as the authors' knowledge goes".

2 Mapping software - PolyWorks

This is the most complex and useful program for mapping that currently exists. This section will cover all the most useful tools and options, as well as dive into the general concept of creating your first Soldat map.

2.1 Terminology

Basic terminology that will be used later:

1. (H), (C), (R), + - (H) - hold the button (and drag the mouse), (C) - click the mentioned button, (R) - release/stop given action, + combine two actions.
2. LMB, RMB, MMB - Left Mouse Button, Right Mouse Button, Middle Mouse button.
3. Polygon - a single triangle that you can place and it will (usually) interact with players. Connecting polygons will allow you to make a map structure/layout.
4. Scenery - images that can be placed in order to enhance visuals of the map.
5. Objects - All kinds of spawn-points, such as flag spawns, player spawns, etc.

2.2 Tools Window

Figure 1: Tools window.

this section describes all **useful** tools and their modifiers. Useless or redundant options won't be mentioned to keep it simple. Some bullet-points require you to also read the chapter about other Windows to fully grasp the context.

1. **General map manipulation**
   (a) MMB (scroll up/down) - zoom in/out.
   (b) MMB (H) - move the map around.
2. **Transform Tool (Q)**
   Allows you to place polygons. This tool is strongly bound to Palette Window.
   (a) RMB (C) - opens a list of polygon types you can choose from.
      i. Normal - basic polygon type.
      ii. Only bullet collide - collides with bullets/grenades only.
      iii. Only player collides - same as *normal*, but doesn't collide with bullets/nades.
      iv. Doesn't collide - doesn't interact with anything, but stays in foreground (in front of the player).
      v. Ice - same as *normal*, but the surface is icy.
      vi. Deadly - kills player on contact.
      vii. Bloody Deadly - same as *deadly*, but adds more blood particles.
      viii. Hurts - inflicts small damage on contact.

ix. Regenerates - regenerates a portion of HP on contact.
x. Lava - same as *hurt*, but inflict flame damage.
xi. Red/Blue/Yellow/Green bullet collide - same as *only bullet collide* but affects only a single team.
xii. Red/Blue/Yellow/Green Player collide - same as *Only player collide* but affects only a single team.
xiii. Bouncy - a bouncy surface. The strength can be adjusted in polygon's property window.
xiv. Explosive - same as *deadly*, but also detonates m79 grenade at the point of contact.
xv. Hurt Flaggers - same as *hurt*, but affects only flag carriers.
xvi. Flagger collides = same as *Only player collides* but affects only flag carriers.
xvii. Non-Flagger collides - same as *Only player collides*, but affects only non-flag carriers.
xviii. Flag collides - interacts/collides only with the flag.
xix. Background transition - same as *doesn't collide* but stays in background (behind the player). Also interacts with *background* polygon type.
xx. Background - same as *Only player collides*, but acts as background if entered from the side where it's connected to *background transition* polygons.
xxi. Textured Quad - Enables fourth LMB (C) action for polygon placement (fourth click creates second polygon connected to the previous one).

(b) LMB (C) x 3 - creates a single triangle polygon. By clicking LMB you choose the position of polygon's vertices.

(c) Shift (H) + LMB (C) x 3 - place polygons, but your mouse will be snapping to round angles (10, 20... degrees, etc.).

3. **Poly Selection (W)**
Allows you to select polygons and scenery. Some tools require a poly/scenery/object to be **selected**, to work properly! You can also combine this type of selection with Vertex Selection (S) tool. This tool is strongly bound to Properties Window.

(a) LMB (C) - select hovered-over polygon or scenery. Clicking it over multiple layers of polygons/scenery will rotate over the selection.

(b) Shift (H) + LMB (C) - add polygon/scenery to the current selection.

(c) Alt (H) + LMB (C) - remove polygon/scenery from current selection.

(d) RMB (C) - opens quick menu for copy/paste/duplicate/arrange of selected items.

4. **Vertex Selection (S)**
Area-oriented variation of a Poly Selection (W) tool. Additionally, it can also select objects. Switching from this tool to Poly Selection (W) will turn all selected vertices into fully selected polygons! This tool is strongly bound to Properties Window.

(a) LMB (C) - if you click on the vertex of a polygon, it will be selected (vertex only).

(b) LMB (H) - drag over polygons/scenery/objects to select them (area effect).

(c) Shift + LMB (C/H) - add items to currently selected by clicking/dragging.

(d) Alt + LMB (C/H) - remove items/vertices from currently selected by clicking/dragging.

(e) RMB (C) - opens menu for copy/paste/duplicate/arrange of selected items.

5. **Vertex colour (D)**
Allows you to paint each vertex of a polygon individually. If no polygon is selected, it will paint what's currently hovered over. Otherwise it will only work on **selected** vertices. The behavior of this tool is strongly bound to Palette Window.

(a) LMB (C/H) - Paint hovered-over polygon vertices with a selected color.

(b) Shift (H) + LMB (C) - precision mode; you have to be really precise with your click to paint a vertex.

6. **Poly Colour (E)**
   Allows you to paint scenery/polygons. Combine with **selection** to get better effects. The behavior of this tool is strongly bound to Palette Window.

   (a) LMB (C) - colour hovered-over polygons/scenery with a given colour. Compared to Vertex Colour (D) tool, it paints the whole polygon always. This behavior can be altered by **selecting** vertices/scenery beforehand. Under this condition, this tool will only paint selected vertices and scenery.

7. **Transform (A)**
   Allows you to move and resize polygons' vertices/scenery and move objects. The behavior of this tool is strongly bound to Properties Window.

   (a) LMB (H) - drag hovered-over vertices, polygons and scenery. However, it works better if you make the desired polygons/scenery **selected** (with selection tools (S)/(W)).

   (b) Shift (H) + LMB (H) - drag **selected** objects in a straight lines only (vertical or horizontal).

   (c) Ctrl (H) + LMB(H) - resize/scale **selected** items by dragging one of the 8 white boxes that appear around the selected items.

   (d) Alt (H) + LMB(H) - rotate **selected** items by dragging one of the 8 white boxes that appear around the selected items.

   (e) LMB (H) + Arrow keys (C or H) - move polygon/scenery/object by 1 pixel. You have to keep holding LMB, unless you **select** desired polygons/scenery/objects beforehand (advised).

   (f) Shift (H) + LMB (H) + Arrow keys (C or H) - same as above, but moves by 10pixel each time.

8. **Texture (F)**
   Allows you to move a texture on polygons. This tool is strongly bound to Map Preferences Window.

   (a) LMB (H) - drag over vertex to stretch the texture near that vertex. If u want to move the texture without distortion, select a whole polygon. Can move texture on multiple vertices/polygons if **selected** beforehand.

   (b) Shift (H) + LMB (H) - Same as above, but drags the texture in straight lines only (vertical and horizontal).

9. **Colour Picker (X or default ".")**
   Lets you pick a color from given polygon/scenery. This tool is strongly bound to Palette Window.

   (a) LMB (C) - clicking on a scenery/vertex will change your current color in the Palette Window to the color of scenery/vertex.

   (b) Shift (H) + LMB (C) - pixel color picker. It will give you the actual pixel color, wherever you click.

10. **Scenery (R)**
    Allows you to place scenery on a map. Some features of this tool are connected to the Scenery Window and Palette Window. In Scenery Window you can choose front/back/middle position of a scenery. In Palette you can choose opacity of the scenery and it's colour. This tool is strongly bound to Properties Window, Palette Window and Scenery Window.

    (a) RMB (C) - Opens a list (soldat/scenery-gfx directory) of available scenery.

    (b) LMB (C x1/x2/x3) - place a scenery. If rotation/scale is enabled in Scenery Window, second and third click will be needed to lock in rotation and scale.

11. **Objects (T)**
    Allows you to place object such as spawns/flags/medikits/grenades/machine-guns. Collider object's size is determined by Palette Window's Radius combo box.

    (a) RMB (C) - Opens a list of available spawn types.

    (b) LMB (C) - Place selected object.

    (c) Delete Key - delete **selected** objects. Use Vertex Selection (S) tool.

2.3 Main Window

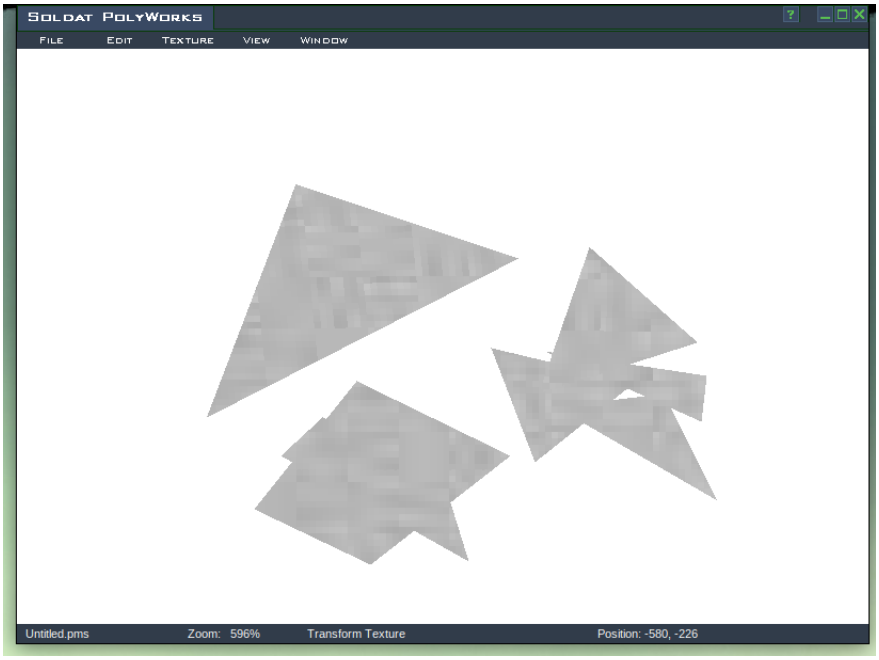

Figure 2: Main Window.

This is where you can place all polygons/scenery/objects in order to create a map. You can manually adjust zoom in the bottom bar.

2.4 Palette Window

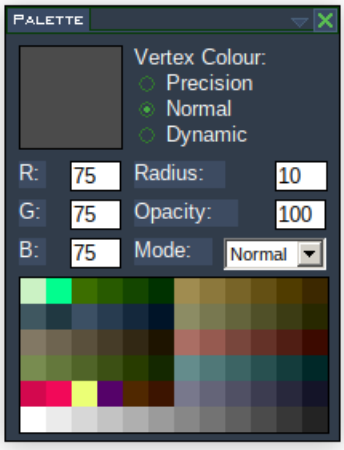

Figure 3: Palette Window.

Here you can choose color that will be applied on new scenery and polygons, as well as their opacity or minor stuff such as radius or coloring mode precision.

1. RMB (C) on the small color boxes - allows you to add currently chosen color (big box) to the palette for later use.

2. LMB (C) on the big color box - opens color picker window, where you can select a specific colour.
3. Opacity - affects the opacity of new polygons and scenery that will be placed. Also affects colour-oriented tools, ex. 10% opacity will make a paint tool 10x weaker (warning: 0% will make painting actions have no effect at all!).
4. Radius - affects the size of collider objects and the radius of Vertex colour (D) tool.

2.5 Properties Window

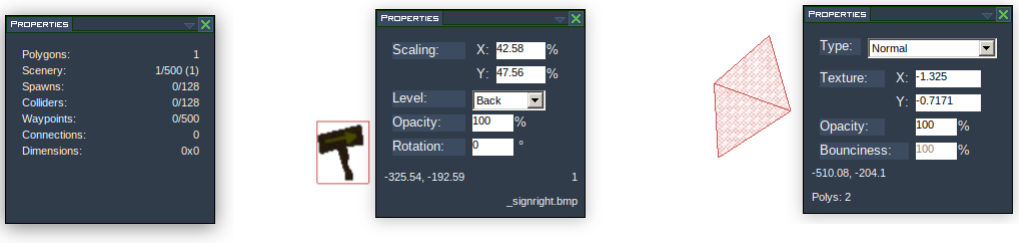

Figure 4: Properties window. From left to right: nothing selected, scenery selected, polygon(s) selected.

This window shows properties of currently **selected** scenery/polygons. This tool allows you to edit properties of **already existing** scenery/polygons. The information in this window changes based on selected items.

1. Nothing selected - general information about the map.
2. Scenery selected:
   (a) Scaling - X/Y scaling in percentages.
   (b) Level - position of the scenery compared to other objects. The scenery of the same position level can be arranged by using Arrange options available under Main Window → Edit.
      i. Back - The scenery will appear behind the polygons, players and scenery of higher level.
      ii. Middle - The scenery will appear in front of the players, but behind the polygons and scenery of higher level.
      iii. Front - The scenery will appear in front of polygons and players, as well as scenery of lower level.
   (c) Opacity - changes the opacity of the selected scenery. If more than one image is selected, this box will display the value of only on of them.
   (d) Rotation - rotates the scenery by degrees specified.
   (e) Right lower corner - displays the name of the selected scenery. If multiple selected, shows the number of selected scenery.
3. Polygon(s) selected:
   (a) Type - The type of the selected polygon(s). If multiple polygons are selected, this box will display only one type. Changing the value of this box affects all selected polygons.
   (b) Texture - The coordinates of the texture on the polygon. Barely useful.
   (c) Bounciness - Active when bouncy polygons are selected. Changing the value of this box affects all selected polygons. Generally, the bigger the value, the stronger the bounce.
   (d) Coordinates - X/Y coordinates of the selected polygon. If only one polygon is selected, additionally displays the ID of the polygon.
   (e) Poly - The number of the polygons currently selected.

Additionally, selecting polygon(s) and switching to Transform (A) tool also affects the content of the Property Window. In this mode, the user can manually enter the X/Y scaling percentage values and rotation in degrees.

2.6 Scenery Window

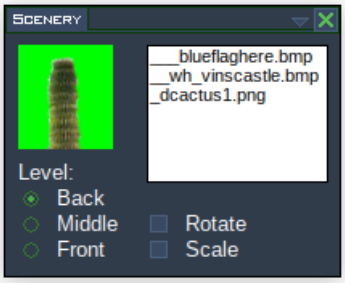

Figure 5: Scenery Window.

Allows you to twist scenery properties. These options do not affect scenery that are already placed on the map. To alter existing scenery, use Property Window.

1. Scenery list - the list of already used scenery. Use to quickly switch to desired scenery when Scenery (R) tool is active.
2. Level - position of the scenery compared to other objects. The scenery of the same position level can be arranged by using Arrange options available under Main Window → Edit.
   (a) Back - The scenery will appear behind the polygons, players and scenery of higher level.
   (b) Middle - The scenery will appear in front of the players, but behind the polygons and scenery of higher level.
   (c) Front - The scenery will appear in front of polygons and players, as well as scenery of lower level.
3. Rotate - Disables/enables rotation when placing new scenery.
4. Scale - Disables/enables scaling when placing new scenery.

2.7 Display Window

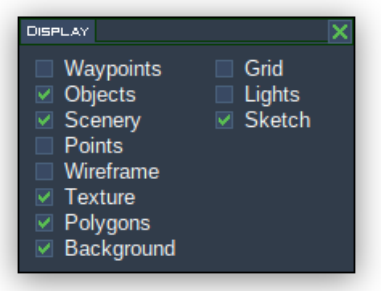

Figure 6: Display Window.

Useful options you can tick at any time:

1. Waypoints - show/hide waypoints for bots.
2. Objects - show/hide objects, such as spawn-points and colliders.

3. Scenery - show/hide scenery.
4. Points - show boxes on top of polygons' vertices.
5. Wireframe - outline polygons and scenery.
6. Texture - show/hide texture on polygons.
7. Polygons - show/hide polygons.
8. Background - show/hide current background color.
9. Grid - show/hide grid (snapping can be turned off/on separately).

2.8 TopBar

Figure 7: TopBar.

Apart from saving/loading maps and properties, TopBar also has many options useful while mapping.

1. Edit → Undo - Undo the last operation. Doesn't work on background color changes.
2. Edit → Redo - Redo the last undo operation.
3. Edit → Copy - copy current selection of items.
4. Edit → Paste - Paste what you copied.
5. Edit → Select All - select all that's currently visible (**ticked in** Display Window).
6. Edit → Invert Selection - Inverts current selection.
7. Edit → Select by Colour - Selects by color chosen in Palette window.
8. Edit → Arrange - changes the relation of selected items compared to other.
9. Edit → Join Vertices - connects selected vertices.
10. Edit → Transform - rotate options for selected items, and flipping horizontally/vertically.
11. Texture → Fix texture - restore texture on polygons (removes texture stretching/translation/scale effects on polygons).
12. Texture → Untexture - removes texture from selected polygons (to be more precise, it does something similar to averaging it's color and removing the pattern).
13. View → snap to vertices - enable/disable vertex snapping.
14. View → snap to grid - enable snapping to grid (if enabled).

2.9 Map Preferences Window.

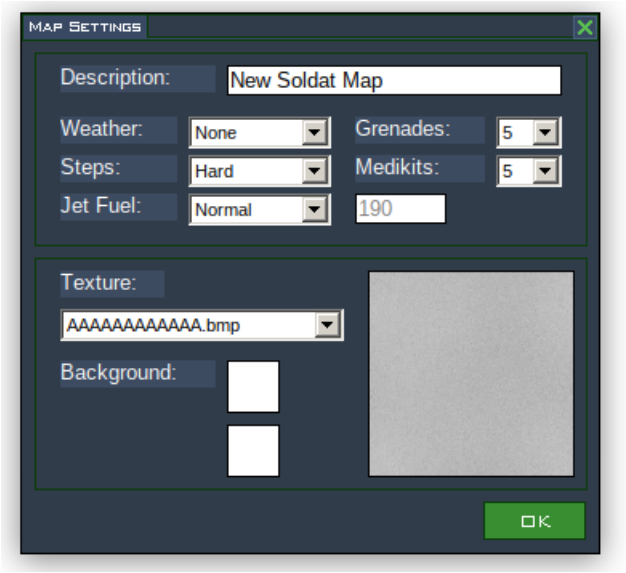

Figure 8: Map Preferences Window.

You can access this window via Topbar → Map Settings or Ctrl + M. Available options:

1. Description - map description that pops up in the chat at the start of the match.
2. Weather - weather effect to be aplied on the map.
3. Steps - audio effects to be applied on the map.
4. Jet Fuel - amount of jet fuel available. Predefined or custom value.
5. Grenades - amount of grenades on the map/per player (or something). Works in single player, overwritten by server settings. Medikits - amount of medikits on the map (or something). Texture - texture to be applied on polygons. Only one file can be used per map. Background - The gradient color of the map background.

2.10 Preferences Window.

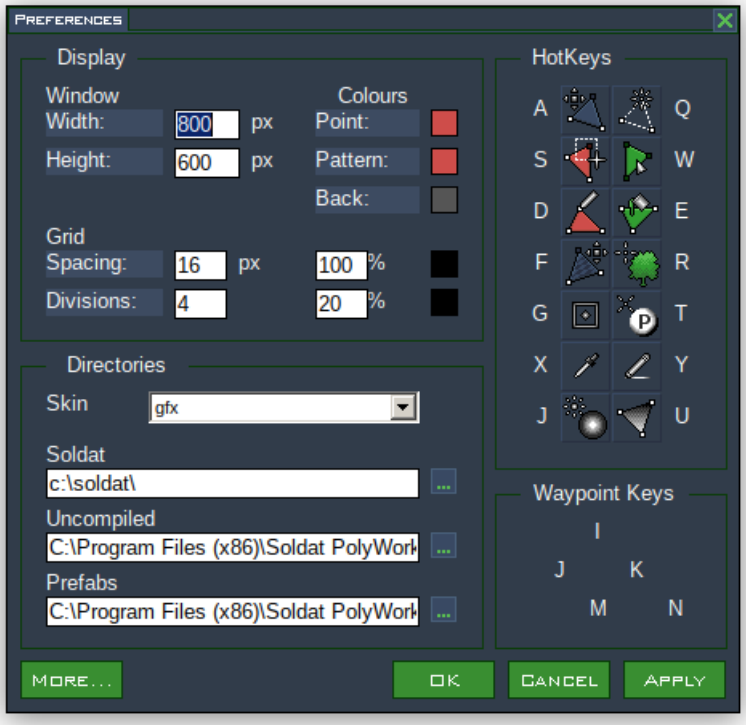

Figure 9: Preferences Window.

Generally speaking, this window contains global settings, not per-map options. You can adjust these according to your liking or follow my guidelines (strongly advised):

1. Window - Current Main Window's dimensions. Recommended: full-screen.
2. Colours - No idea. Recommended: default.
3. Grid - Grid dimensions. Recommended: Spacing 16px, Divisions: 4, other: default. You can also change the colors of the grid here.
4. Directories - Soldat directories:
   (a) Skin - the path to Polywork's UI graphics. Recommended: the fanciest you can lay your hands upon, or default.
   (b) Soldat - The main Soldat directory. Recommended: Your path to Soldat installation folder.
   (c) Uncompiled - Directory to store uncompiled maps at. Recommended: default.
   (d) Prefabs - Directory with Polyworks prefabs. Recommended: default.
   (e) More... - Advanced options for blending of the wireframe/selection colors around polygons etc. Recommended: default. Also an options to enable 4 points of interaction for scenery (easier to select/manipulate scenery, but might clutter the space). Recommended: default.
   (f) HotKeys - hot-keys for all Tools. Recommended: default + color picker to "X".

3 Good practices

Keep in mind these are general rules and there is a bunch of reasons not to follow them, some of them will be covered in the following sections. There are a lot of great maps that break some of these rules - the key is to also use your brain when doing it.

Screenshots use common polygon colors: normal - white, deadly - red, ice - blue, green - invisible.

1. **Clean polygon structure**
   Clean structure will greatly lower a chance of polybugs appearing. All of it's components are listed below:

   (a) Use as few polygons, as possible. Creating a square takes 2 polygons, not 10 or 20. What might be even more surprising for beginners - triangle can be created using only one polygon. [10]

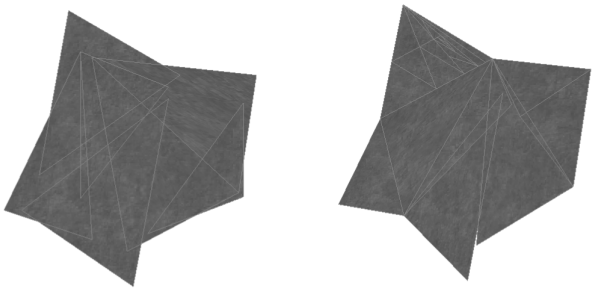

Figure 10: Bad structure on the right, good structure on the left.

   If you want to create a complex structure (for better shading possibilities, etc.) use *doesn't collide* polygons, and keep the colliding foundation as simple as possible. [11], [[10]]

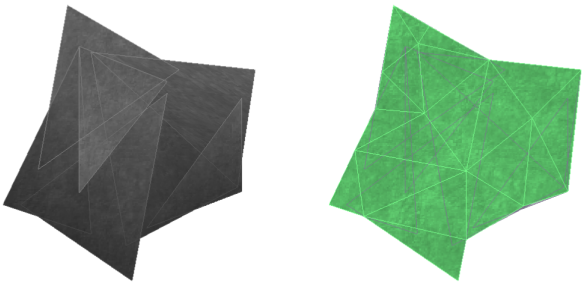

Figure 11: Bad shading on a good structure, good structure before shading on the right. Notice the green *doesn't collide* polygons overlay on the good base structure.

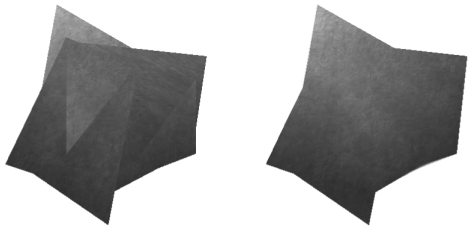

Figure 12: Shading results comparison.

(b) Avoid exposed polygon edges and polygon connections. Fewer of them minimizes the chance of polybugs occurring.

(c) Do not overlap polygons unless your goal is to fix existing polybugs (explained in detail in *Polybugs* section).

(d) Don't use very thin polygons (+/- thinner than soldier's torso). The bigger the polygon, the easier it is to prevent it from affecting a player in an odd, buggy way.

2. **Colors and contrast**
   Try to use eye friendly colors, that are easy on the eyes. Using consistent color scheme for different polygon types will make it more bearable for players.

3. **Visuals**
   Those are okay, as long as the scenery/polygons don't cover up too much. The Inverse Scenery Difficulty Principle goes as follows: The more crucial the part of a map is, the more simpler the visuals, to the point of not having any scenery at all. One well placed scenery will make it more pretty to the eye and won't cripple the gameplay!

4. **Naming convention**
   Use "ctf_" + "(optional) creator's initials/tag_" + "map name". Examples: ctf_mapper_mapname, ctf_randomname. It's also a good idea to keep all the letters lowercase. It's also popular to make the first letter in the map name uppercase: ctf_Examplemap. If u plan on using your own tag, download a mappack beforehand and check if your tag isn't already used by someone with similar nickname.

   Add your full nickname in the map's description, so they won't become anonymous as the time passes.

5. **Consistent difficulty level (climb)**
   Pay attention to the difficulty level of each separate jump or combo. If your goal is to make a noob-friendly map, it might be a good idea to avoid complex, pixel perfect combos. On the other side, you don't want to put 10 repetitive W-spike-jumps in the middle of a speedrun, mechanic-heavy map, just to make it irritating.

6. **Memory/fake maps (climb)**
   Don't create fake platforms that you'll fall through, or platforms that are safe to jump on, interspersed with deadly platforms that look exactly like the safe ones. Once players have memorized the jumps for those, they are really just gimmicky and less than useless.

7. **Shortcuts (climb)**
   Avoid creating shortcuts that cut a lot of time out of otherwise long climb maps. Once discovered (usually in map editors) they become just another really short maps.

   Creating a good shortcut doesn't mean hiding it behind a fake wall, but rather giving a player a high risk/high reward path, where beating a difficult obstacle is rewarded with a shorter cap time.

8. **Spawn points**
   Make sure the player spawn doesn't randomly put players into polygons or kills them. Put it noticeably far from both normal and deadly polygons.

Flag spawn points should be placed in areas, where they don't move or randomly spawn in different locations (unless there is an idea behind it). Using invisible flag-collide polygons is your friend.

(climb) If you want to separate blue and red flag spawns, consider how likely it is to find yourself in a situation, where you reached the flag spawn, and someone already took the flag (ruined your run).

(climb) Don't use more than one alpha/bravo spawn per map, unless it's crucial for the map to be captured. One spawn will allow speedrunners to have consistent times and fair challenge.

4 Polybugs

Knowing how to fix polybugs is just as important as knowing where to look for them! This knowledge also helps when creating jumps based around exploiting them (explained in different section). This section also assumes you maintain clean polygon structure.

It's worth noting, that sometimes applying more than one *possible fix* will give better results, so it's recommended to try mixing them together and seeing what works best.

1. **Corner bounce**
   Occurs when using *non-deadly* polygons in a right, isosceles triangle shape, usually smaller ones. The enemy will most likely hide in the right angle of the poly in question. Ice polygons tend to be less vulnerable, but it's not always the case.

   Possible fixes:

   (a) Use different polygon shape, focus on removing the right angle.

   (b) Place deadly polygon near the troubling corner.

   (c) Place floor above the polygon, making exploiting of the bug impossible/useless.

   (d) Cover the corner with deadly polygon.

   (e) Make the poly ice type.

2. **Sticky wall**
   Occurs on all vertical *non-deadly/non-ice* walls. Player can get glued to the wall when falling and hugging it, losing all momentum.

   Possible fixes:

   (a) Tilt the wall slightly, so that it's a bit overhanging. It also lowers or removes the possibility of grabbing the lower corner of the wall (sometimes more or less tricky, but often possible).

   (b) Make the wall *ice*. It might turn into climbable wall instead.

   (c) Make the wall significantly wider. It will most likely turn into climbable wall instead.

3. **Climbable/bouncy wall**
   Occurs on all vertical *non-deadly* walls. Player can fall a bit into the wall and then gets bounced off of it. Good timing allows players to climb/jump on these walls.

   Possible fixes:

   (a) Tilt the wall slightly, so that it's a bit overhanging. It also lowers or removes the possibility of grabbing the lower corner of the wall (sometimes more or less tricky, but often possible).

   (b) Make the wall significantly thinner. It will most likely turn into sticky wall.

4. **Landing on polygons causing bounce**
   Player landing on poly gets stuck in/bounced off the poly. More likely to occur when landing with high momentum.

   Possible fixes:

   (a) Make polygon thicker.

(b) Move polygons edges as far from the landing zone as possible.

(c) Transform the polygon in a way that makes player more likely to land on the thicker part of the polygon.

5. **Magnets**
   You know this badboy. Usually ping and netcode will play the biggest part here, but you can try to minimize it.

   Possible fixes:

   (a) Don't connect player's platforms with *deadly* polygons near places where the player is expected to jump.

   (b) Avoid forcing player to move towards *deadly* polygons in very tight places.

6. **Polygons connections causing bounces**
   Occurs when player steps/jumps on the connection of two or more polygons.

   Possible fixes:

   (a) Remove the connection, using the power of your brain.

   (b) Rearrange the polygons, so that the player running from one side to the other starts from the polygon set the furthest to front, ending on the polygon moved the furthest back.

   (c) Cover the edge with (*invisible*) polygon.

   (d) Overlap the polygons, eliminating the connection.

   (e) Add a deadly obstacle just-so-happened to cover the troubling part.

   (f) Change the whole structure and give up (seriously, sometimes there's nothing you can do).

Friendly tips: You might wonder how to fix all of these issues and still have a somewhat good looking map. The answer is simple and for once, very easy. **Use *invisible* and *doesn't collide* polygons!**

You want thin polygon, but it bounces? Turn it into a *doesn't collide* polygon and place huge, invisible platform underneath! I don't want to tilt this wall, because i want 90° angles everywhere? Turn it into *doesn't collide* and tilt the *invisible, colliding* wall behind it! No one will notice 10 pixel difference when playing.

5 Jumps (climb)

This section gathers all useful information regarding creation of smooth, uncheesable jumps and more fancy movement setups. Even though creating such jumps is achieved mostly by a lot of in-game testing, it often starts with the same core.

Most importantly, most of the jumps can be either a part of a combo, meaning you require player's momentum from previous jump(s) or static, meaning you expect a player to reset all momentum before the obstacle/don't require any preserved momentum. Therefore core setups presented below will mainly refer to static jumps since taking players momentum into account is map specific and require individual testing.

Speed check mentioned in the text below is a jump following, leading or the discussed jump where a player needs certain preserved momentum to perform it successfully.

This part also assumes you already internalised the content of *Good practices* and *Polybugs* sections.

1. **Cannonball**
   Forcing a player to do a cannonball requires:

   (a) Taking into account, whether the player already has any horizontal momentum, or is capable of getting it from the starting platform.

   If the goal is to make a static cannonball jump, it would be a good idea to also force a player to stop before the jump (resetting his movement speed right before it). Going for a combo cannonball means moving the landing platform further and lower in comparison to the static one.

(b) The easiest way to prevent player from cheesing static cannonball using *lateflip* is adding a low-hanging ceiling. If you are afraid of cheesing your combo cannonball with long *lateflip* jump, adding a simple speed check on the end of a combo can solve the problem (long late flip is the simplest option). Otherwise moving the landing platform further down and low will eventually make the combo uncheesable even without the need of a speed check.

2. **Reversed (late) cannonball**
   If you want to force a player to do a *reversed (late) cannonball* there is no reliable way of doing that without forcing a static position or using some sort of a guide.

   (a) Notice *lateflip* and *latecannonball* are different in only two points - the moment you perform a flip (*lateflip's* flip is high, losing more momentum, *latecannonball's* flip is flat but gives more momentum). The goal is to either kill player on the flip animation or on the cannonball's speed check.

   (b) You can also use *normal* polygon roof as a guide, to force player into the exact right spot. Pay even more attention to cannonball's speed check.

3. **Semi cannonball**
   Shares the same issues as basic *canonball*. On top of that:

   (a) Placing a *deadly* obstacle between the starting and landing platforms with appropriate height is crucial. Focus mostly on the height near both of the platform, the middle isn't that important and can leave room for *lateflip* cheese.

4. **Booster / 1 pixel jumps**
   It's possible to jump on polygons which have no width or/and height (straight line or point). Jumping on a line is possible on both of it's ends (equivalent to two points).

   (a) Since you can't see them in-game, it requires some sort of indicators (scenery, *doesn't collide* polygons).

   (b) Stacking a bunch of these point-polygons in a straight line will give a feeling of a booster. You don't need to make it very dense.

   (c) Falling/running parallel to the booster line will make it harder to go through it the more momentum you have.

   (d) You can use 1 pixel polygons to fake *walljumps*, make reliable *corner jumps*, *corner grabs* and other stuff! Be creative!

5. **Walljumps**
   A wide spectrum of jumps, basically all platforms which a player can jump off of, but staying on them is very hard or close to impossible.

   (a) Choosing a wall angle should be dictated by the players' possible ways of approach and their speed. Generally straight vertical wall is a good choice, however using different angles almost always gives more consistent results and fluid movement.

   (b) If possible, use *ice* polygons.

   (c) Use polygons that are significantly wider than the walljumping side. This rule is even more important when using walls close to vertical or when not using *ice* polygons.

   (d) Doing a *wallgrab* (jumping into the direction of the wall when you are on it) can give random results, such as breaking the jump animation, if the wall is vertical or catching the lower edge of the *walljump*. Solve it by adding a stopping wall below and paying attention to player's momentum.

6. **Corner bounce/jumps/grabs**
   Even though very similar to *walljumps*, corner variations focus on using only the very corner of the poly, not the entire wall.

   (a) Corner grabs/jumps are very precise, but work on every polygon edge. To make it more reliable, you can use *1 pixel jumps* to make it more consistent.

(b) corner bounce works best with a right, isosceles triangle-shaped polygon. Usually quite random and give results similar or worse to bounce polygons. The pictures do show some more reliable setups.

7. **Nade jumps**
   Wtf dude.

8. **Combos**
   Creating good combo jumps is utilizing every piece of information contained in this guide. It means connecting polybug free, polished jumps together and decorating it with appropriate, gameplay-friendly visuals.

   (a) Check each separate jump for mentioned polybugs and make it uncheesable using the knownledge aquired while reading this guide.

   (b) Make sure every piece of combo cannot be completed without the preserved momentum acquired from the starting jump.

   (c) Place every platform in a spot that allows the combo to be completed flawlessly, without quirky movement if it has been *perfectly* executed. From there add as much error margin as you want, usually by making the platforms wider, but still paying attention to speed checks.

   The bigger the error margin, the easier the combo. A fun combo would essentially mean forcing only a chain of jumps, giving broad freedom of movement once in the air. Hard combo would also push a player to pay attention to their movement while mid-air, jumping positions and landing, often going hand in hand with narrow deadly corridors.

   (d) It's close to impossible to make a combo (or even a jump) completely polybug free, therefore it's not a bright idea to expect players to execute it perfectly. Keeping this in mind, even the hardest combos should give just enough room for error, to compensate for the game imperfection.

   (e) It's always great to test the whole combo from start to finish but sometimes it could be very tedious or nearly impossible. However, most of the combos, especially the longer ones, have stopping/peak points - points where player's momentum resets. This is where you can place testing spawn points, and try individual parts of the combo.

6 General concept

This section assumes you have at least read the previous chapters and have at least a faint idea of what tools you can use in order to create your first Soldat map. To have a general understanding of what's ahead of you, this section briefly explains the most important steps.

1. Prepare your environment.

   (a) Place custom-made textures in Soldat dir /textures and custom-made scenery in /scenery-gfx. Make sure the file extensions are supported.

   (b) Enable visibility of all windows under Window → Show All.

   (c) Edit your Edit → Preferences... according to the guide.

   (d) Edit your Edit → Map Settings... according to the guide.

   (e) Make sure everything is set correctly in Display Window.

   (f) File → Save as... your new project.

   (g) File → Compile as... your new project under the same name. It is required to always compile the map before trying to run it.

2. Create polygon structure.

   (a) Create the most bare-bone structure to test your concept and layout.

   (b) Place objects necessary for in-game testing (spawns, etc.).

   (c) Test in-game and use Good Practices to fix problems and create bug-free, final polygon structure.

3. Enhance the visuals.

   (a) Use shading and painting tools to add some color to your polygon structure.

   (b) Use scenery/background transition polygons to add some more life to your map.

   (c) Test in-game to fix visual aspects.

4. Place objects.

   (a) Place spawns for playable teams, flags, grenades etc.

   (b) Test in-game to catch placement bugs, adjust for better game-play experience.

5. Repeat the steps above until satisfied with the result.

6. Pack your compiled map with Soldat Map Packer or do it manually. Make sure to include used scenery and texture. Everything can be found in Soldat and Soldat Polyworks installation directories.

7. Release your map. Don't forget to attach a screenshot.