Ahex Notes

Introduction

Ahex is a minimalistic puzzle game with hexagonal movements. This document lists thoughts and ideas.

Aesthetic

- [x] I'd like to go for a soft pastel aesthetic, inspired by games such as Tunic.
- [x] The world has a baseline made of water, which is implemented using the `bevy_water` crate.
- [x] Tiles are hexagonal with the tip pointing up. This is a stylistic design choice. The hexagon is defined through a circumcircle with unit radius, yielding the following geometry:

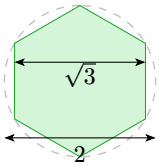

Figure 1: Geometry of a hex tile

Due to Bevy, the *xz*-plane is the "flat" ground plane. The *y* axis points up. Hence, Figure 1 uses the *x* and *z* axes.

Mechanics

- [x] The camera can rotate, so that the player can see behind tall objects.
- [x] The camera should rotate only in intervals of $\frac{\pi}{3}$ radians at a time, so that the hexagons always end up looking the same. Of course this transition should be fluent.
- [x] The controls (`W/E/A/D/Z/X`) should adapt based on the angle of the camera. Otherwise controls are too confusing for the player if the camera is rotated.
- [x] The player should be able to restart a level using some button, remote from the usual controls. It could be a combination like `Ctrl+R`. Currently: `Backspace`.
  - [ ] If a level has become unwinnable (due to the player or the *star* falling into the water), the game should hint to use this restart combination.
- [ ] The player should be able to undo his last moves with `R` or similar. To do this, we need to keep the state of the entire level for each step.
  - [ ] The player should be able to undo multiple moves as well.

Puzzle ideas

- [x] The objective is a *star*. Upon collecting the *star*, the level is completed.
  - [ ] Levels may be replayable by implementing a secondary *star* after completion. I'm not sure yet if I want to do this for every level.
  - [ ] A *star* is subject to physics just like the player. Hence, a *star* can fall down or be lost to the abyss.
- [x] The player can fall *down*, but not jump *up*. This causes a significant asymmetry for the *y* axis.

- [ ] The player has a certain height. This disallows him from squeezing between two tiles (one above the other) if there isn't enough height left.
- [x] Tiles can be programmed to move along a *path*. This can have multiple sub-variants:
  - [x] Back and forth between two coordinates: this is useful for simple elevators (going up and down the *y* axis) or short hops to form bridges.
  - [x] A line segment: an extension of just moving back and forth; specify a direction and an amplitude.
  - [x] A circle: to recreate floating platforms that can take the player to multiple places, or to even simulate conveyor belts.
  - [x] The full solution: a directional path, i.e. a `Vec<(isize, isize, isize)>`. This would allow a tile to move multiple coordinates in one step, as well as take any arbitrary path. Of course this path *should* return to the tile's original position, although this is not a strict requirement.
  - [x] If the player is on a tile that is moving, the player should move along with it.
    - [ ] This should keep in mind collisions, e.g. the player can be shoved off it it hits a wall along the way.
- [ ] Tiles may be *slippery*. If the player moves on them, the player will continue to move until an end is reached (wall, or edge of the map).
- [ ] Tiles may be *fragile*. After the player has stepped on it, it will crumble as soon as the player steps off it.
  - [ ] Some *fragile* tiles might be rechargable.
- [ ] *Crates* are solid objects that the player can't traverse through, but can push. A push is only possible if the crate can occupy the target hex.
  - [ ] *Crates* could come in two variants: small hex and full hex. A full crate occupies the entire ground of the tile that it is on. These big crates cannot squeeze through pairs of pillars (like the pillbug in the game Hive!
- [ ] The player is only strong enough to push one *crate* at a time (I think). A series of crates are therefore not pushable in the direction that they form a series in.
- [ ] *Lasers* block the player from moving through them, much like walls. Lasers extend across the entire level, until blocked by something solid.
  - [ ] *Lasers* may be blocked by the player pushing a *crate* into its path.
- [ ] *Trampolines* cause the player or any other solid object to jump one tile. This can be used to cross bridges.
  - [ ] *Trampolines* could come in fixed or in *crate*-like variants (which can be moved). Note that for this, tile heights must be uniform!
  - [ ] If a player falls down flat on top of a *trampoline*, they can no longer move in any direction. This should trigger the restart hint.
  - [ ] Jumps should keep into account collisions. The jump might be canceled halfway if the player would otherwise hit a wall. This could cause the player to fall down early.

Level file format

- [x] The format will be TOML. This is because it allows comments, is not indent-sensitive, has sensible types, and is supported by the `serde` crate.
- [x] The format should be easily extendible. Everything should start in a section to allow for extension.

- [x] Most levels will need only one layer of tiles (i.e. at most one tile at a given y value). However, the format should allow for multiple layers in case a level will contain caves, stacked layers, and so on.
- [x] The height map can be rectangular, and should map to corresponding xz-coordinates. The values of tiles can range from `0` to `9` by default. In practice tiles probably won't get higher than this.
- [x] Tiles can be applied one or multiple sets of *modifiers*. Modifiers include:
  - [x] Has a player on top of it
  - [x] Has a goal on top of it
  - [ ] Is slippery
  - [ ] Is fragile
  - [ ] Has a crate on top of it
  - [ ] Is a trampoline