# Physicalized Game Interfaces

*Final Design Document*

## Overview

Classical game interfaces, such as keyboards, mice, and gamepads, all force the user to pre-interpret their input and “digitize” it so it can be digested by a computer. As computer vision has progressed, however, it has become more and more possible for computers to interpret inexact input from the physical world. It is this premise that formed the foundation for this exploration of physical computing possibilities.

## Core Goals

1. Create a mapping between physical space and a digital game world
2. Bring the game world back into the physical input space
3. Create a method for users to interact with physical space that works intuitively as an input into the game world mapping
4. Bundle the above aspects into a free, open source application, paired with an inexpensive and portable physical setup

This phsyical-to-digital mapping is the real power of a physicalized game interface, as it not only allows players to intuit the controls of the game diegetically from the game’s setup, but also gives a sense of immediacy, impact, and efficacy to the players actions.

## Audience

- Game designers
- Installation or interactive experience designers
- Procedural, digital, and tech artists

## User Experience

To set up the project, the following are required:

- A cloned copy of the GitHub repository
- A webcam
- A projector
- Mounting equipment

To begin, the webcam and projector must be mounted such that they are facing the same flat surface, and the webcam’s view is exactly covered by the projector’s coverage area. Then, after building the application, the user can run it and pass it a command line flag to tell it which webcam to use. Once the program starts, a new fullscreen window will appear that can then be moved to display on the projector, after which the application is fully ready to use.

To use the setup, users can place tag16h5-format AprilTags of IDs 0-4 in the webcam view, and different shapes of different colors will be projected onto their location and tracked as they move.

## Code

This project uses Bevy, a Rust-based entity component system (ECS) and game engine. Using bevy, the functionality of the program is broken down into webcam handling, structure processing, and tag tracking modules. The webcam access is achieved through the Nokhwa rust library, and the tag tracking through the AprilTag rust libary, which itself is a port of the C library by the same name.

The control flow of the application is as follows:

1. At startup, several “structure descriptors” are loaded into memory. These are shapes and other objects like sprites that are associated with a tag ID.
2. Also at startup, a global handle to the webcam in use and the current frame from it are established so other parts of the codebase can access the image data.
3. Each frame, a new image is taken from the webcam and loaded into a buffer, which is then read by a detector object to generate a list of possible tag detections, each with an id, a center coordinate, and a certainty value representing the computer’s confidence in the detection.
4. These detections are processed every frame and, if they match a structure, used to display that structure on the input surface.
5. After tags are no longer visible and a cooldown has passed, associated structures are made invisible.

## Challenges

Libraries and dependencies posed a significant challenge during this project. Since the software for this project needs to span graphics/rendering, computer vision, *and* webcam interfacing, there were many dependencies and sub-dependencies required for the final product to be possible. Using the `nix` package manager made finding and plugging in those dependencies much easier, however, and made the development environment much more reproducible.

Most of the issues with libraries were at the compilation stage, and next to no runtime issues were encountered throughout the project due to library functionality. There were significant issues with Nokhwa due to a deprecated sub-dependency, fixed by applying the revision from an open pull request on GitHub as the library version. There were a few other issues with Nokhwa and Bevy on setup, but once the library issues were fixed progress was speedy and relatively painless. In the future, however, it would be prudent to start setting up the libraries and the project structure during the design phase as the scope of the project becomes clearer such that there is more time to pivot, switch libraries, or fix the existing ones.

Figuring out a way to mount the camera and projector was also a challenge, though the members of the Folk Computer project had a very useful and pertinent guide for creating a similar setup, which formed the basis of the setup for this project. Calibration between the camera and projector was eventually

settled to be by hand, since it’s much easier to judge the distance once and lock the mounts than to try to calculate transformations in code.

## Next Steps

At the time of writing, there are only hard-coded “structures,” and there is no way for the end user to add more. The application is also command-line based, and can therefore be intimidating for an inexperienced end user to operate.

Since the use of Bevy’s ECS forces the project to be structured in a modular fashion, however, it would be a simple task to expand the structure loading library to accept configurations from an external object descriptor format like JSON, YAML, or TOML. This expansion would allow users to describe what structures they want tag IDs to be associated with, and upon running the program those tag IDs would be immediately available for use.

Since Bevy also has UI library integrations and supports multiple windows, running the application and configuring it graphically is also within close reach. A more developed version of this product should emphasize usability, since the ultimate goal is to facilitate making physical and interactive experiences and broaden their reach.

## Applications

Future applications of this project could include games, interactive installations, and other tech/art interface uses. Some examples include:

- A maze game where the player has to navigate their character by sliding a tag along the play surface
- An enhanced virtual tabletop gaming setup
- Interactive procedural art using particle systems spawned at the location of a moving tag
- Physicalized 2D painting software

## Images


Figure 1: Preliminary calibration testing, with the camera output displayed on a laptop monitor.

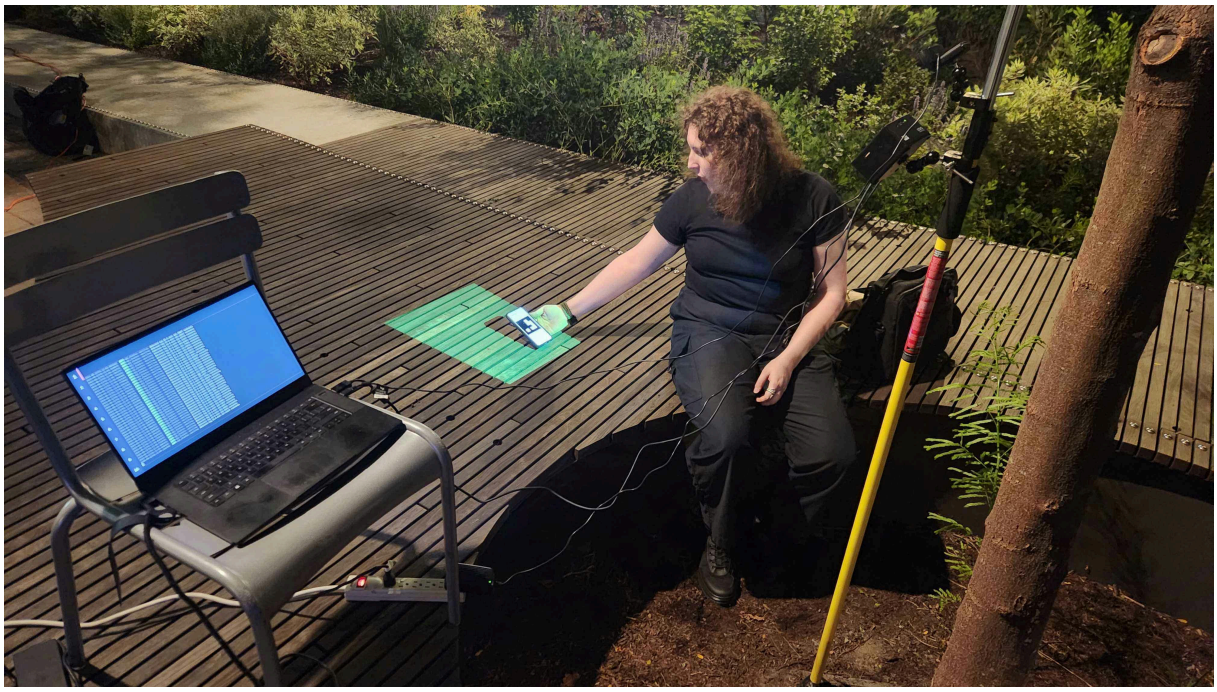

Figure 2: PGI on display at Boston Indies: Demo Night.


Figure 3: A basic paint demo using duplicated shapes to draw lines and tag IDs to control stroke width.

## Inspirations / Sources

- Folk Computers
- Nokhwa
- Bevy
- Nokhwa plugin for Bevy
- AprilTag for Rust
- Basic video playback in Bevy (example)