# Learn & Code 2021.1 “SET” Project

## Intended Audience / User

The intended audience is SET players at In Time Tec.

Other stakeholders include Eddy and Trevor. When appropriate, they will simulate roles such as business analyst, product owner technical architect, technical lead, technical account manager, et cetera.

## MVP

Deadline: By end of Learn & Code.

Scope:

* One player versus computer;
* Web UI.

## MVP: Specifically Excluded

* No authn/authz;
* No console or native UI;
* No leaderboard;
* No multiplayer;
* No saved games;
* No saved scores;
* No statistics or analysis;
* No user data.

## Application Scope: Overall Design

* End-user experience and application delivery: Web UI.
* Multi-user with authentication and authorization.

## Application Scope: Post-MVP Features / Stretch Goals

In order of most important to least:

* Multiplayer;
* Authn and user data;
* Saved scores, saved games, and leaderboard.
* Should we consider an improved computer player?

## Tool Stack

* IDE: Visual Studio (edition TBD)
* Language: C# + .NET 5
* Revision control: GitHub
* Progress tracking: GitHub
* Testing: xUnit.net (?)
* DevOps CI/CD pipeline: out of scope

## Progress Tracking

* Stories
* Acceptance criteria
* Current experience
* Desired experience
* Any blockers?
* Is the story blocking anything?

## Test Strategy

* TDD?
* E2E?
* Manual?
* Other?

Let’s wait until that chapter!

## Eddy’s Algorithms

In our 2021-05-05 class, it became clear that more explanation was needed surrounding Eddy’s bit-manipulation algorithms.

### Cards as Bit Fields

Let’s treat the case of a single card facet, as the various facets are homologous. Shape resides in the low byte, so we’ll use that.

|  |  |  |  |
| --- | --- | --- | --- |
| **Diamond** | **Squiggle** | **Oval** | **Different** |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

Each facet value occupies its own bit. If we wish to think in terms of sequential numbers, we can examine how many bits we have shifted left from the rightmost bit:

|  |  |  |  |
| --- | --- | --- | --- |
| **Diamond** | **Squiggle** | **Oval** | **Different** |
| 3 | . | . | . |
| . | 2 | . | . |
| . | . | 1 | . |
| . | . | . | 0 |

As of the 2021-05-05 class, the enum FacetValue is written in terms of 0b\_0001\_0000 shifted accordingly. Let’s ignore for now that we are working in the high nybble. What matters is the relation of these four bits to one another.

What is the deal with “Different”?! There’s no such card! Very true. In addition to matching “all diamonds” or “all squiggles” or “all ovals”, we can match “all different shapes”. The bit about “different” has something to do with that sort of a match. Think of the above table as assigning sequential numbers to match styles; if we can pull that off, it would be a most convenient representation. (How do we convert between bit position and sequential numbers? We’ll worry about that another time.)

### Combining Cards

What happens if we have three squiggles, then combine with a binary “and” operation?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Combo #1** | **Diamond** | **Squiggle** | **Oval** | **Different** |
| **Card #1: Squiggle** | 0 | 1 | 0 | 0 |
| **Card #2: Squiggle** | 0 | 1 | 0 | 0 |
| **Card #3: Squiggle** | 0 | 1 | 0 | 0 |
| **Combine via binary &** | & | & | & | & |
| **Result** | 0 | 1 | 0 | 0 |

This makes intuitive sense: The logic tells us that all “Squiggle” inputs give us a “Squiggle” output:

* 1 & 1 & 1 = 1

So far, so good. What about a failed match? Let’s try two squiggles and a diamond:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Combo #2** | **Diamond** | **Squiggle** | **Oval** | **Different** |
| **Card #1: Squiggle** | 0 | 1 | 0 | 0 |
| **Card #2: Squiggle** | 0 | 1 | 0 | 0 |
| **Card #3: Diamond** | 1 | 0 | 0 | 0 |
| **Combine via binary &** | & | & | & | & |
| **Result** | 0 | 0 | 0 | 0 |

No match. The “Diamond” column shows that:

* 0 & 0 & 1 = 0

and the “Squiggle” column informs us that:

* 1 & 1 & 0 = 0

Great! So far, we’re getting exactly what we need! Let’s push our luck, try each card being different, and see what happens!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Combo #3** | **Diamond** | **Squiggle** | **Oval** | **Different** |
| **Card #1: Oval** | 0 | 0 | 1 | 0 |
| **Card #2: Diamond** | 1 | 0 | 0 | 0 |
| **Card #3: Squiggle** | 0 | 1 | 0 | 0 |
| **Combine via binary &** | & | & | & | & |
| **Result** | 0 | 0 | 0 | 0 |

Uh-oh. Looks just like “no match”:

* Diamond: 0 & 1 & 0 = 0
* Squiggle: 0 & 0 & 1 = 0
* Oval: 1 & 0 & 0 = 0

Is there anything that we can do to fix this – short of resorting to enumeration or loops?

### Opposite Day

At this point, we are all out of options… ***not!*** We might as well just use a loop or enumeration… ***not!***

What would happen if we were to change zeroes to ones, and vice versa? Intuitively speaking, this would quit telling us what a card is, and tell us what a card is ***not***. Is there a name for this unary bitwise operation? Yup. It’s called “not”.

### Combos on Opposite Day

What happens if we have three squiggles, then combine with a binary “and” operation?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Combo #1** | **Not Diamond** | **Not Squiggle** | **Not Oval** | **Something** |
| **Card #1: Squiggle (Not Diamond and Not Oval)** | 1 | 0 | 1 | X |
| **Card #2: Squiggle (Not Diamond and Not Oval)** | 1 | 0 | 1 | X |
| **Card #3: Squiggle (Not Diamond and Not Oval)** | 1 | 0 | 1 | X |
| **Combine via binary &** | & | & | & | & |
| **Result** | 1 | 0 | 1 | X |

This makes intuitive sense: The logic tells us that all “Squiggle (Not Diamond and Not Oval)” inputs give us a “Squiggle (Not Diamond and Not Oval)” output:

* Diamond: 1 & 1 & 1 = 1
* Squiggle: 0 & 0 & 0 = 0
* Oval: 1 & 1 & 1 = 1

(Ignore the “Something” column. The “X” actually means “don’t care”; we simply aren’t using that column, and will make no assumptions about what it contains.)

So far, so good. What about a failed match? Let’s try two squiggles and a diamond:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Combo #2** | **Not Diamond** | **Not Squiggle** | **Not Oval** | **Something** |
| **Card #1: Squiggle (Not Diamond and Not Oval)** | 1 | 0 | 1 | X |
| **Card #2: Squiggle (Not Diamond and Not Oval)** | 1 | 0 | 1 | X |
| **Card #3: Diamond (Not Squiggle and Not Oval)** | 0 | 1 | 1 | X |
| **Combine via binary &** | & | & | & | & |
| **Result** | 0 | 0 | 1 | X |

What is each column telling us?

* Not Diamond: 1 & 1 & 0 = 0 (not all cards are Not Diamond, so at least some are Diamond)
* Not Squiggle: 0 & 0 & 1 = 0 (not all cards are Not Squiggle, so at least some are Squiggle)
* Not Oval: 1 & 1 & 1 = 1 (all cards are Not Oval)

Sensible, yes. Useful? Doesn’t really seem to be yet. Whether we have an all-same match or a non-match, we end up with some bits that are set, and some that are not.

Great! So far, we’re getting exactly what we need! Let’s push our luck, try each card being different, and see what happens!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Combo #3** | **Not Diamond** | **Not Squiggle** | **Not Oval** | **Something** |
| **Card #1: Oval (Not Diamond and Not Squiggle)** | 1 | 1 | 0 | X |
| **Card #2: Diamond (Not Squiggle and Not Oval)** | 0 | 1 | 1 | X |
| **Card #3: Squiggle (Not Diamond and Not Oval)** | 1 | 0 | 1 | X |
| **Combine via binary &** | & | & | & | & |
| **Result** | 0 | 0 | 0 | X |

Something magical just happened! All the columns (you *are* ignoring “Something”, right?!) just became zero:

* Not Diamond: 1 & 0 & 1 = 0
* Not Squiggle: 1 & 1 & 0 = 0
* Not Oval: 0 & 1 & 1 = 0

Note that we have three cards, and three facet-value columns. The *only* way to have a zero in each facet-value column is for all three cards to be of different facet values. Let’s see what happens if we have our uninverted values in the high nybble, force “Different” to one, and have our inverted values in the low nybble:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Combo #3** | **Diamond** | **Squiggle** | **Oval** | **Different** | **Not Diamond** | **Not Squiggle** | **Not Oval** | **Something** |
| **Card #1: Oval (Not Diamond and Not Squiggle)** | 0 | 0 | 1 | 1 | 1 | 1 | 0 | X |
| **Card #2: Diamond (Not Squiggle and Not Oval)** | 1 | 0 | 0 | 1 | 0 | 1 | 1 | X |
| **Card #3: Squiggle (Not Diamond and Not Oval)** | 0 | 1 | 0 | 1 | 1 | 0 | 1 | X |
| **Combine via binary &** | & | & | & | & | & | & | & | & |
| **Result** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | X |

Yay! The result is correct! There is just one nagging problem: We *forced* “Different” to be 1. Let’s see what we can do with a little subtraction:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Combo #3** | **Diamond** | **Squiggle** | **Oval** | **Different** | **Not Diamond** | **Not Squiggle** | **Not Oval** | **Something** |
| **From Above** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | X |
| **Magic Delta** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **Subtraction** | - | - | - | - | - | - | - | - |
| **Result** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | X |

Ignore the facet-value columns to the right of “Different”. We don’t care about those any more, and will mark these columns with “X” in a moment. What matters is that our subtraction borrowed all the way through to the “Different” column… and that the *only* way for this to happen is when “Not Diamond”, “Not Squiggle”, and “Not Oval” all are zero. We now have “Different” indicating the opposite of different; a quick binary “not” (implementing via exclusive-or so we can flip the bit that matters) fixes this:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Combo #3** | **Diamond** | **Squiggle** | **Oval** | **Different** | **Not Diamond** | **Not Squiggle** | **Not Oval** | **Something** |
| **From Above** | 0 | 0 | 0 | 0 | X | X | X | X |
| **XOR Mask** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **Combine via binary ^** | ^ | ^ | ^ | ^ | ^ | ^ | ^ | ^ |
| **Result** | 0 | 0 | 0 | 1 | X | X | X | X |

Hooray! The result is correct – for all combinations of facet values! Try it and see!