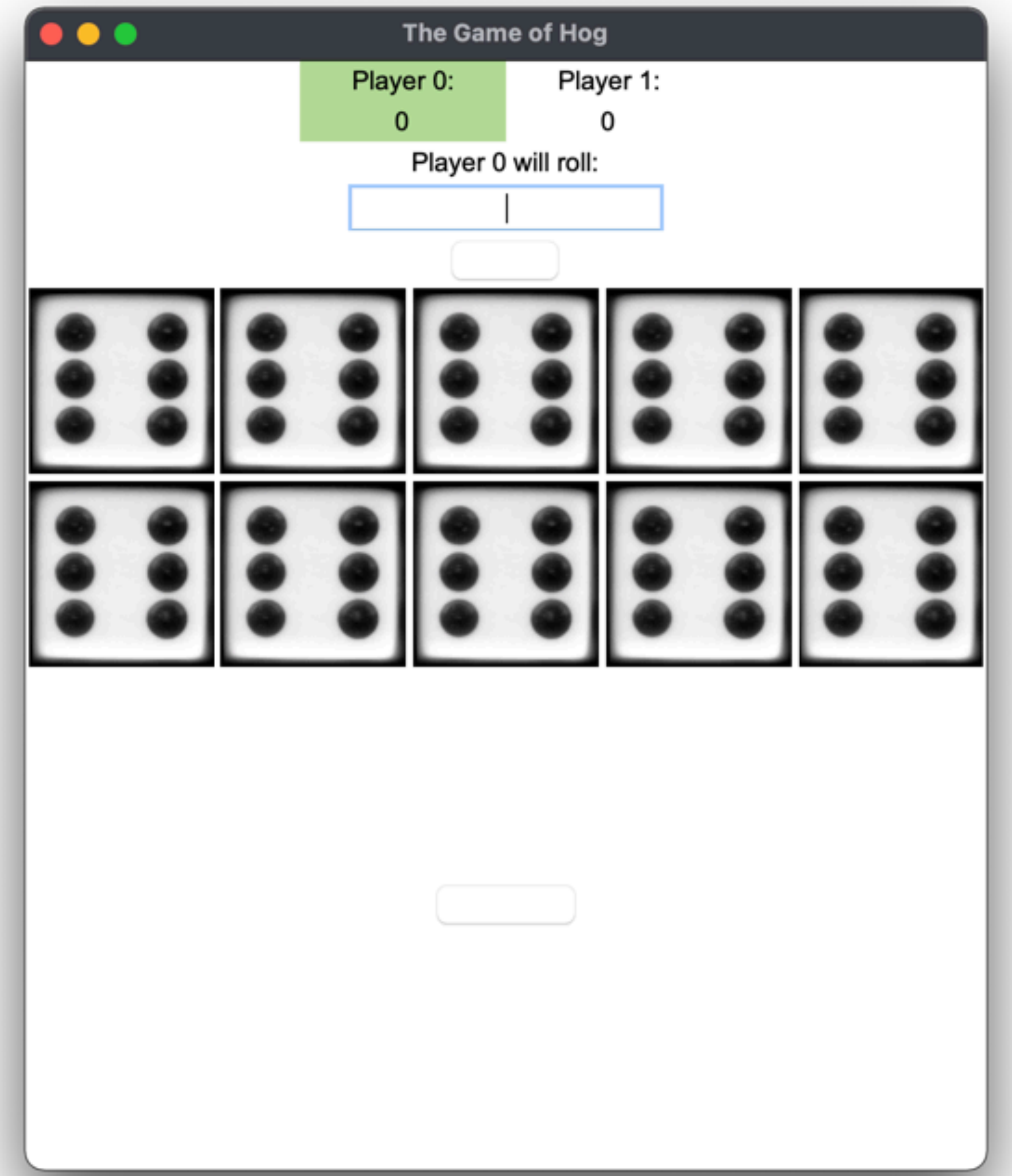
Projects

Class and Individual Projects

Winston Purnomo

Hog

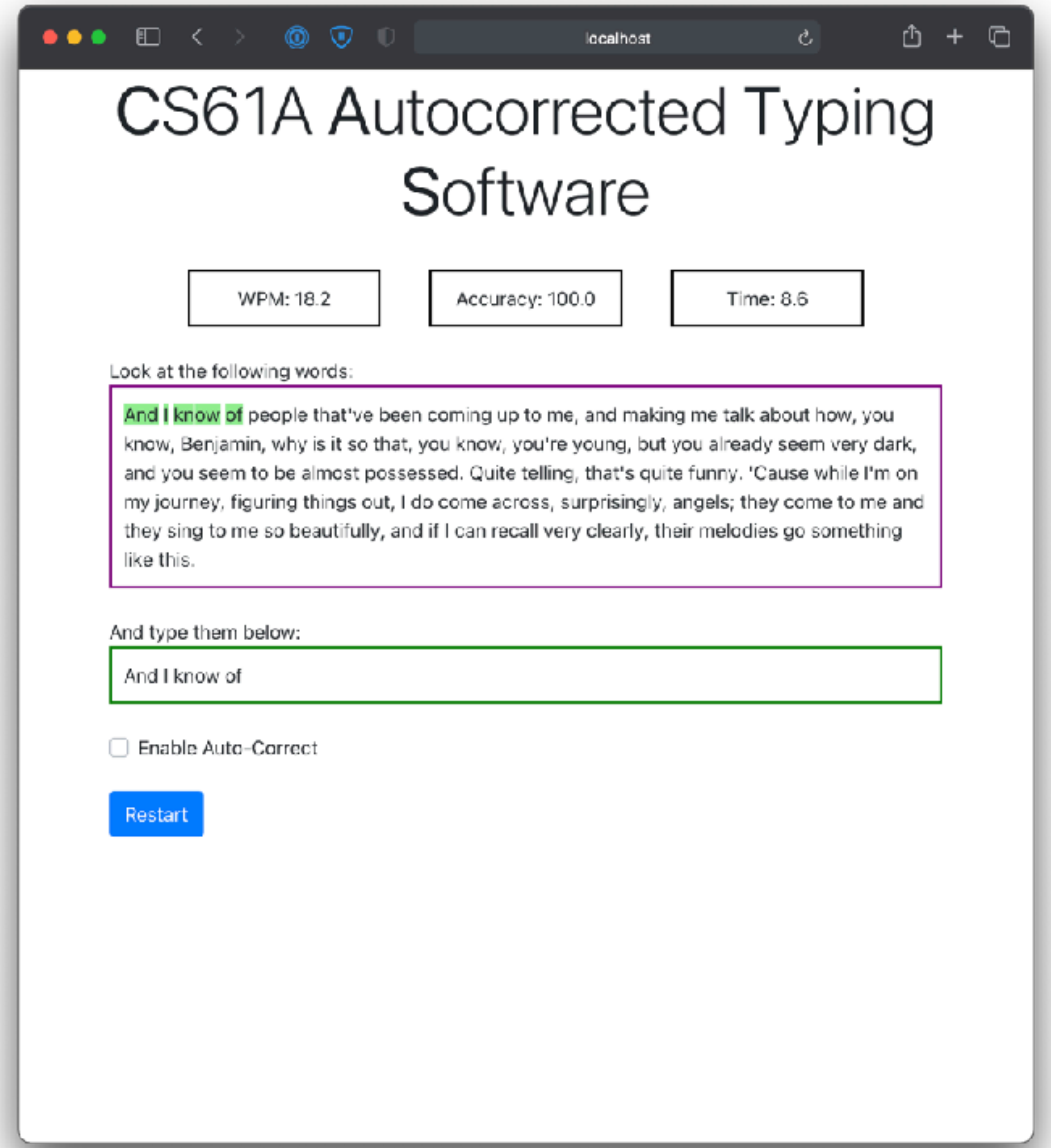
CS 61A: The Structure and Interpretation of Computer Programs



- Hog is a dice strategy game where two players alternate turns trying to be the first to end a turn with at least 100 total points.
- On each turn, the current player chooses some number of dice to roll, up to 10, but there are various ways to gain and lose points in the rules.
- In this project, we learnt use of **functions**, **logical expressions**, and **higher-order functions**.

Cats

CS 61A: The Structure and Interpretation of Computer Programs



CS61A Autocorrected Typing Software

WPM: 18.2 | Accuracy: 100.0 | Time: 8.6

Look at the following words:

And I know of people that've been coming up to me, and making me talk about how, you know, Benjamin, why is it so that, you know, you're young, but you already seem very dark, and you seem to be almost possessed. Quite telling, that's quite funny. 'Cause while I'm on my journey, figuring things out, I do come across, surprisingly, angels; they come to me and they sing to me so beautifully, and if I can recall very clearly, their melodies go something like this.

And type them below:

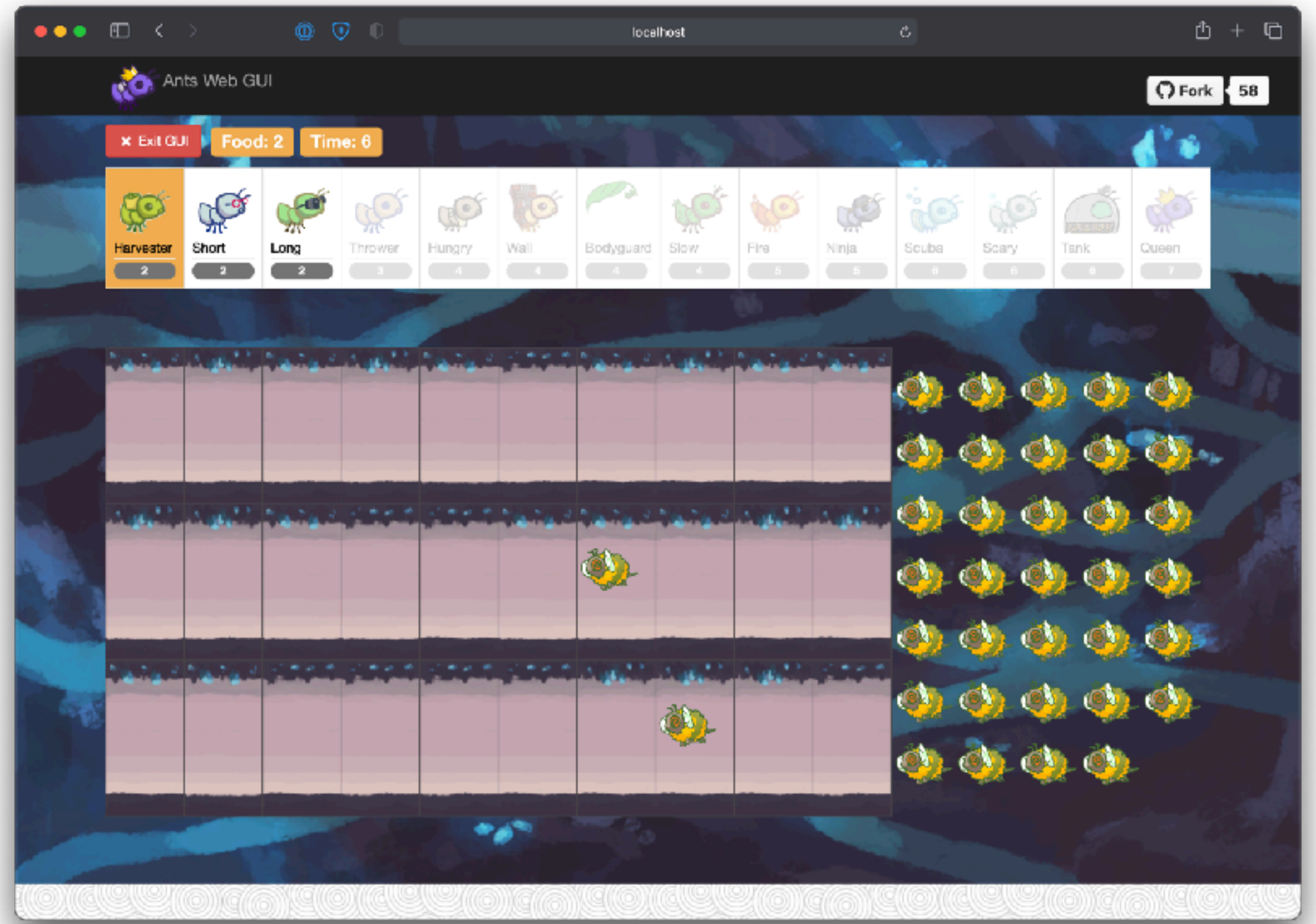
And I know of

Enable Auto-Correct

- Cats is a typing game to measure the accuracy and speed of your typing.
- You are given a passage to type and you will be measured on two measures — accuracy, and words per minute.
- In this project, we used the concepts of **recursion**, **data abstraction** and **mutation**.

Ants

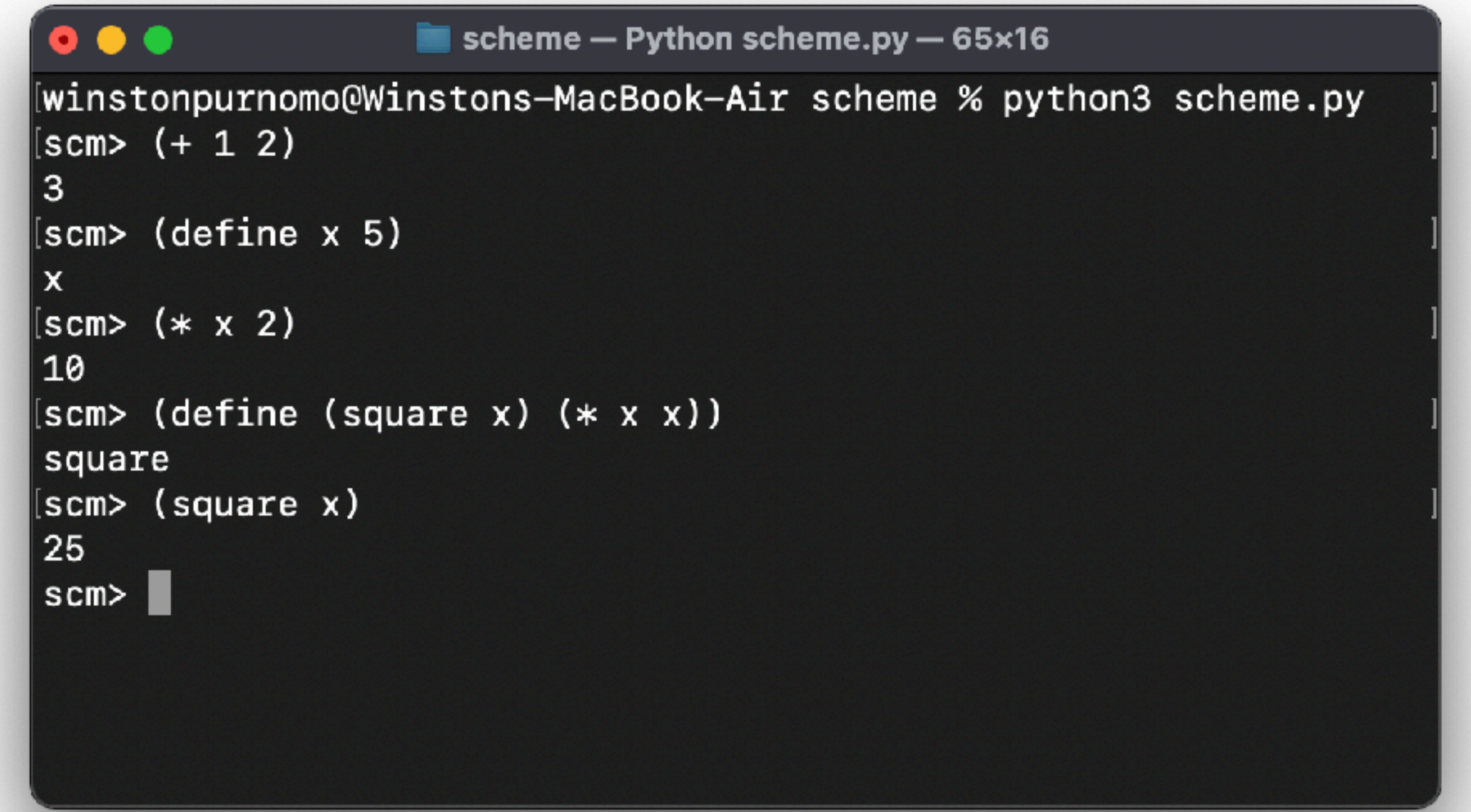
CS 61A: The Structure and Interpretation of Computer Programs



- Ants is a tower defense game inspired by Plants vs. Zombies.
- The game consists of a series of turns, where bees enter an ant colony, and the ants must defend the colony.
- This project explored the concept of **object-oriented programming**.

Scheme

CS 61A: The Structure and Interpretation of Computer Programs

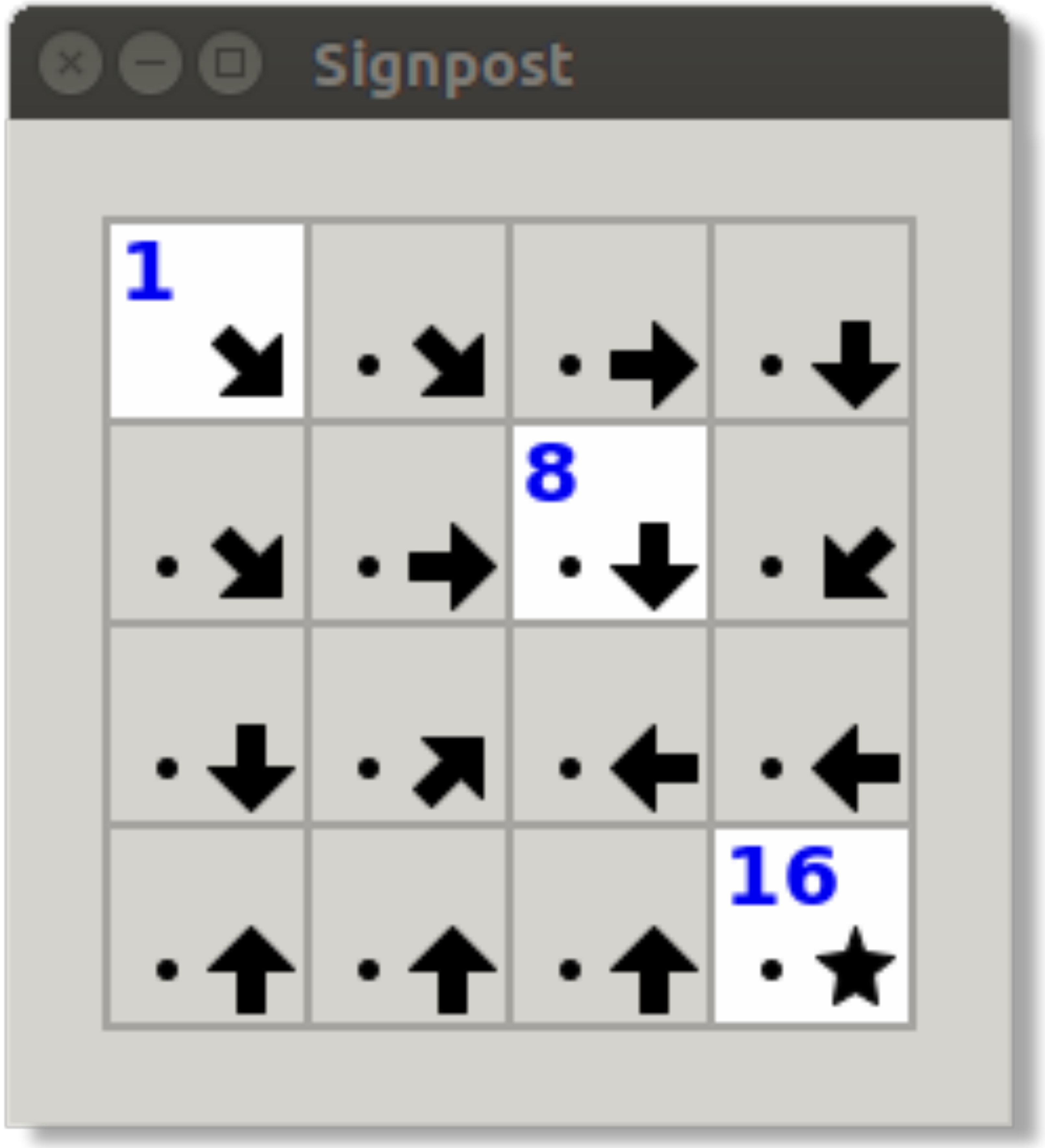


```
winstonpurnomo@Winstons-MacBook-Air scheme % python3 scheme.py
scm> (+ 1 2)
3
scm> (define x 5)
x
scm> (* x 2)
10
scm> (define (square x) (* x x))
square
scm> (square x)
25
scm>
```

- In this project, we were asked to build a Scheme interpreter written in Python.
- Scheme is a Lisp-based programming language.
- In this project, we implemented our knowledge of **interpreters** and **functional programming**.

Signpost

CS 61B: Data Structures



- Signpost is based on a mini collection of GUI games from Simon Tatham.
- It is played on a rectangular grid of cells with arrows pointing in one of eight directions, and the idea is to connect the squares, in the direction they are pointing, into a sequence traversing the entire board.
- The project is meant as an initial introduction to **Java** and **JUnit Testing**.

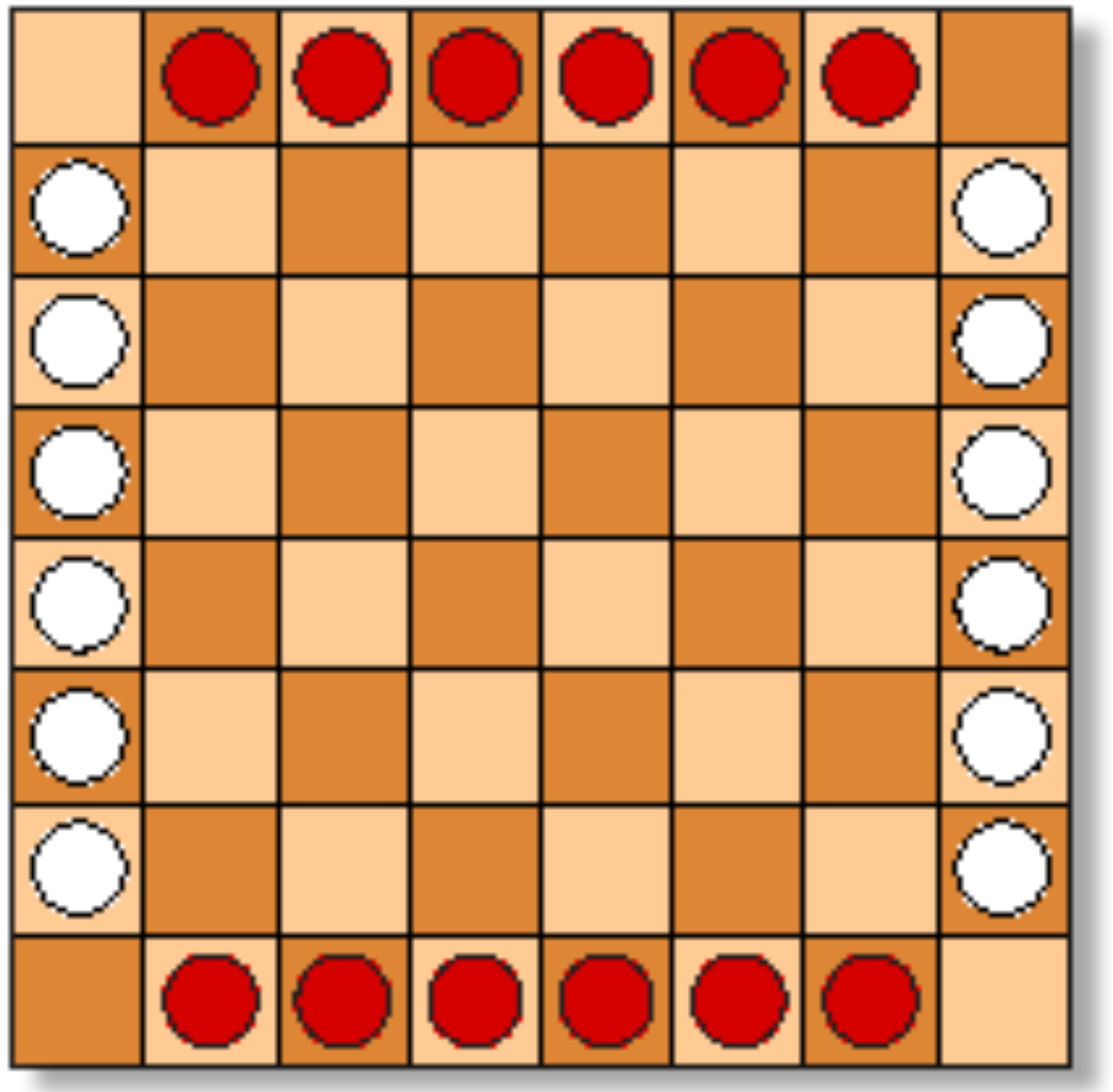
Enigma

CS 61B: Data Structures

- The Enigma Machine was a World War II-era crypto tool, used by the Nazi Germans, to send encrypted text based on a rotor system.
- In this project, we were asked to build an Enigma Machine simulator that will take a message and setting as input, and it will produce a cipher text identical to an actual machine.
- We developed our skills in **object-oriented programming**, **interfaces** and **access variables**.

Lines of Action

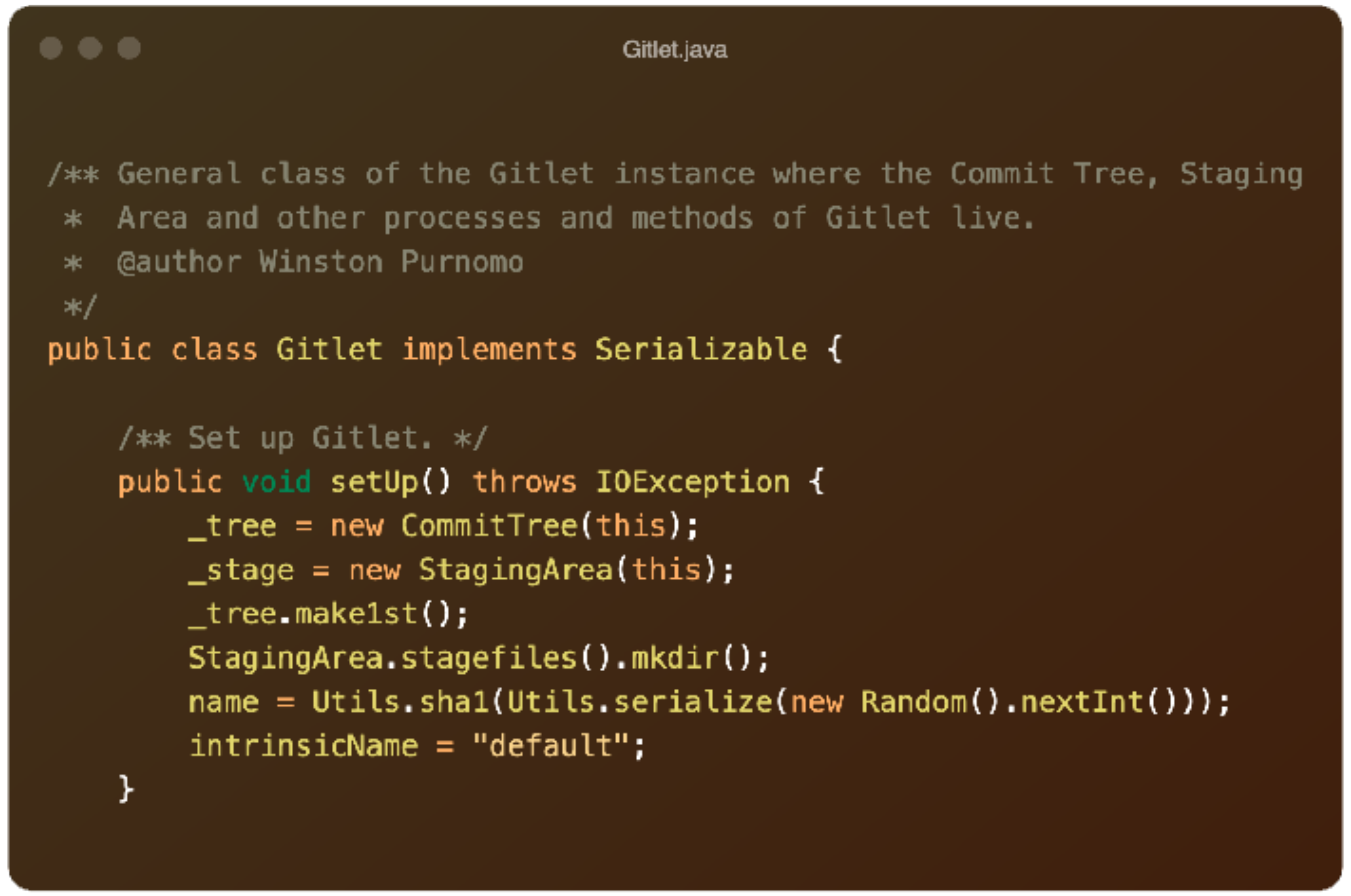
CS 61B: Data Structures



- Lines of Action is a board game by Claude Soucie, played on a checkerboard with ordinary checkers pieces.
- Players take turns, each moving a piece and possibly capturing an opponent piece. The goal is to get all of one's pieces into a connected group.
- In this project, we learnt **complexity**, **tree-searching** and **game trees**.

Gitlet

CS 61B: Data Structures

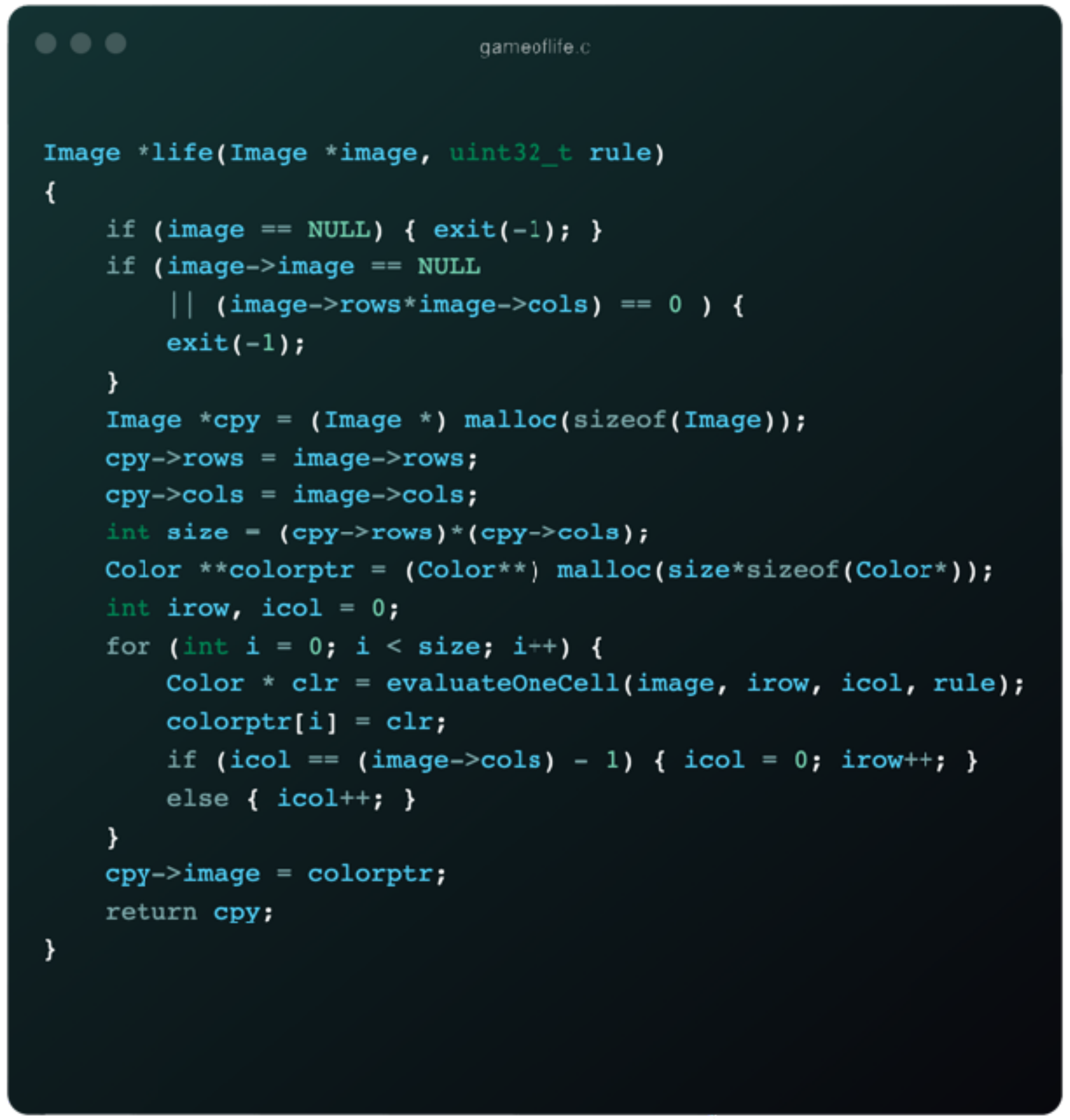

Gitlet.java

```java
/** General class of the Gitlet instance where the Commit Tree, Staging
 *  Area and other processes and methods of Gitlet live.
 *  @author Winston Purnomo
 */
public class Gitlet implements Serializable {

    /** Set up Gitlet. */
    public void setUp() throws IOException {
        _tree = new CommitTree(this);
        _stage = new StagingArea(this);
        _tree.make1st();
        StagingArea.stagefiles().mkdir();
        name = Utils.sha1(Utils.serialize(new Random().nextInt()));
        intrinsicName = "default";
    }
```

- Gitlet is a version-control system which mimics some of the basic features of Git.
- Users can add and commit files, find logs, checkout older versions, create new branches, as well as merge branches.
- This project featured no starter code, and required us to have an innate understanding of Git's inner workings, **complexity**, **hashing**, **file management**, and **graphs**.

Conway's Game of Life

CS 61C: Great Ideas in Computer Architecture

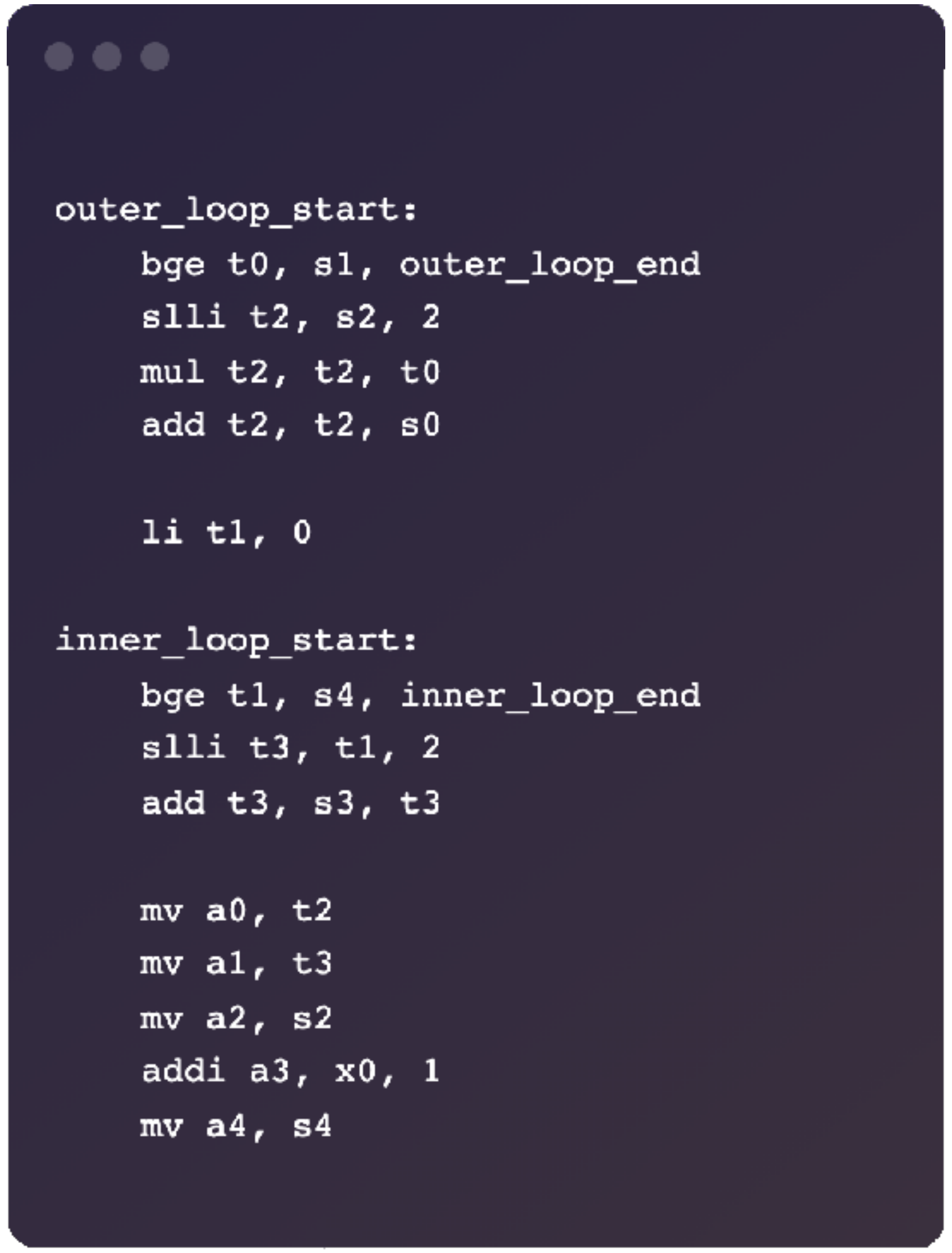


```c
Image *life(Image *image, uint32_t rule)
{
    if (image == NULL) { exit(-1); }
    if (image->image == NULL
        || (image->rows*image->cols) == 0 ) {
        exit(-1);
    }
    Image *cpy = (Image *) malloc(sizeof(Image));
    cpy->rows = image->rows;
    cpy->cols = image->cols;
    int size = (cpy->rows)*(cpy->cols);
    Color **colorptr = (Color**) malloc(size*sizeof(Color*));
    int irow, icol = 0;
    for (int i = 0; i < size; i++) {
        Color * clr = evaluateOneCell(image, irow, icol, rule);
        colorptr[i] = clr;
        if (icol == (image->cols) - 1) { icol = 0; irow++; }
        else { icol++; }
    }
    cpy->image = colorptr;
    return cpy;
}
```

- Conway's Game of Life is a "zero-player" game, which plays automatically given an initial state.
- In this project, we worked with **handling image files**, **pointers** and **structs**, serving as a general **introduction to C**.

CS61Classify

CS 61C: Great Ideas in Computer Architecture

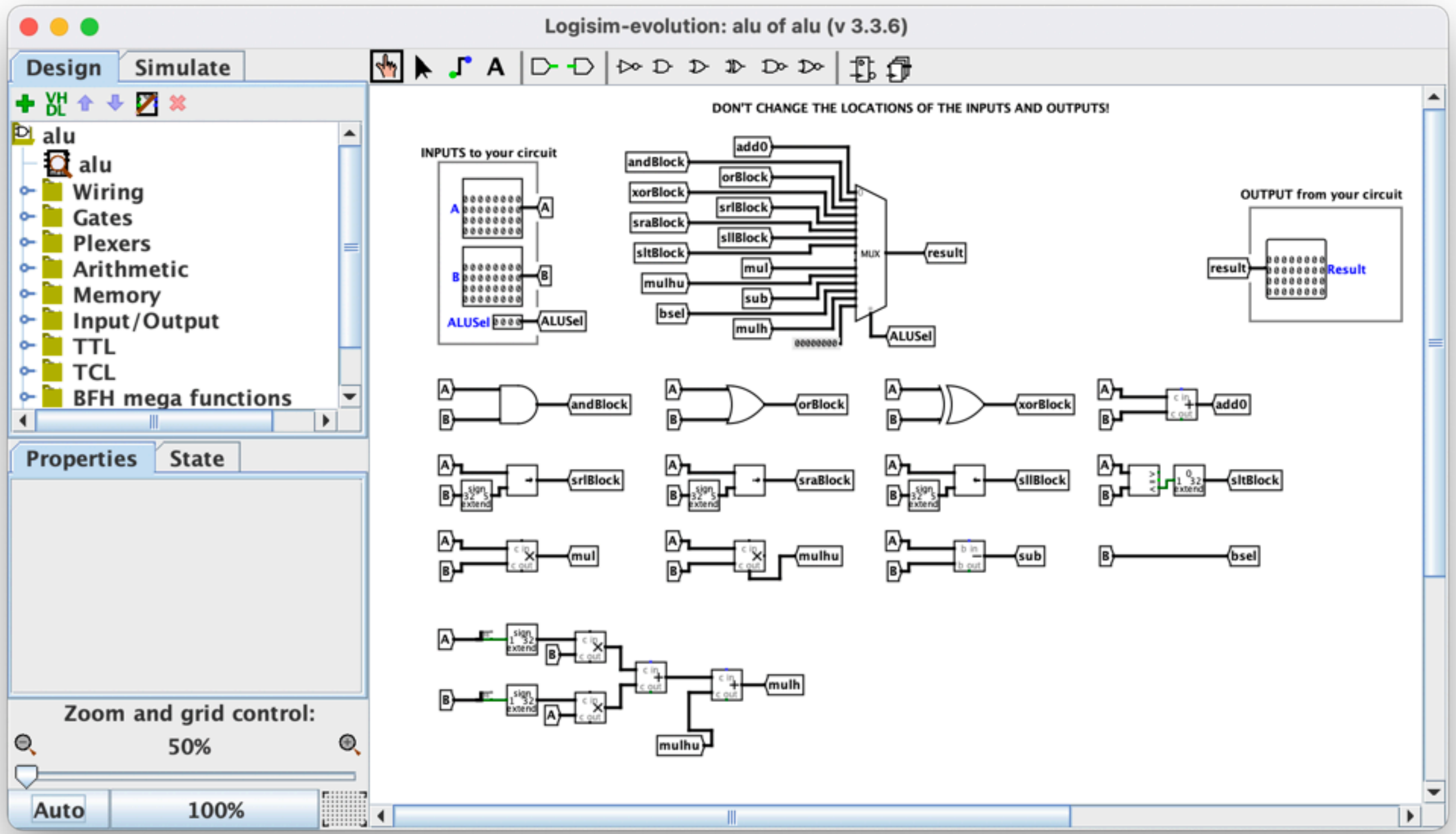


```
outer_loop_start:
    bge t0, s1, outer_loop_end
    slli t2, s2, 2
    mul t2, t2, t0
    add t2, t2, s0

    li t1, 0

inner_loop_start:
    bge t1, s4, inner_loop_end
    slli t3, t1, 2
    add t3, s3, t3

    mv a0, t2
    mv a1, t3
    mv a2, s2
    addi a3, x0, 1
    mv a4, s4
```

- In this project, we wrote a simple Artificial Neural Network using the RISC-V assembly language.
- This project served as a general introduction to **RISC-V** and **assembly language**, including **calling conventions** and **CALL**.

CS61CPU

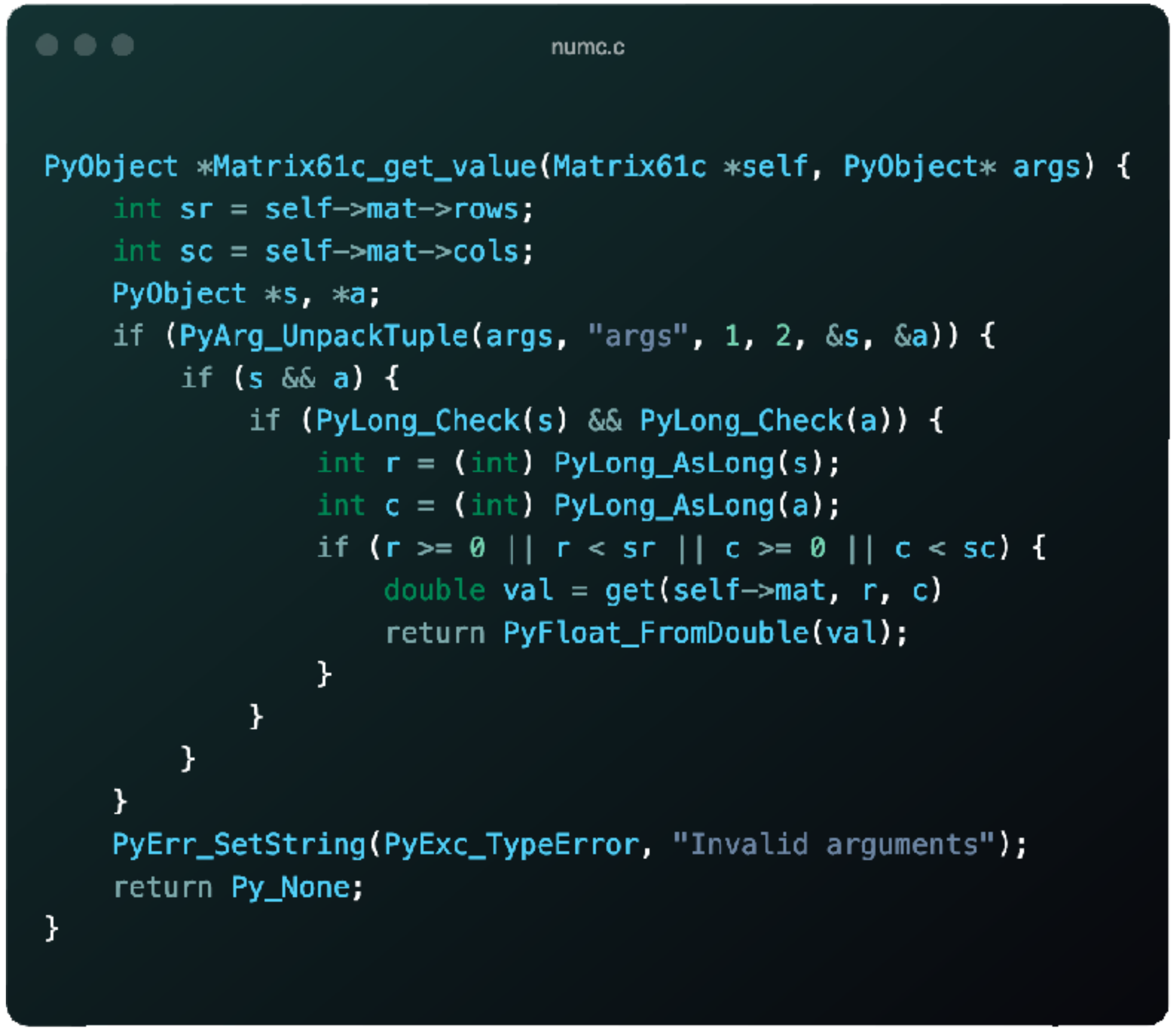
CS 61C: Great Ideas in Computer Architecture



- In this project, we were asked with wiring up the ALU, RegFile and datapath for a RISC-V CPU using the Logisim software.
- This CPU is able to run RISC-V instructions, such as those written in the previous project.
- We used knowledge of the RISC-V **datapath** and **pipeline** to complete this project.

NumC

CS 61C: Great Ideas in Computer Architecture



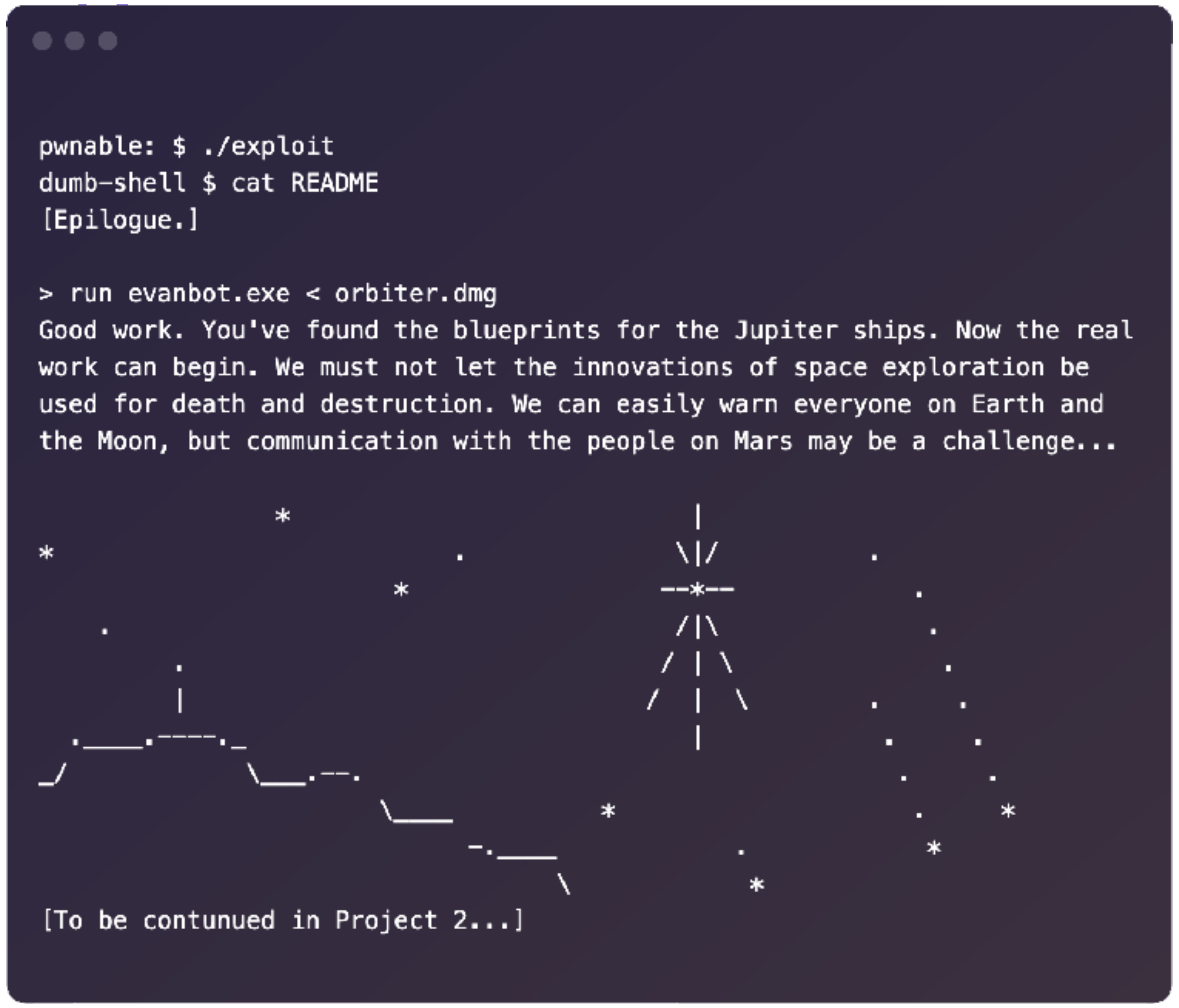
numc.c

```
PyObject *Matrix61c_get_value(Matrix61c *self, PyObject* args) {
    int sr = self->mat->rows;
    int sc = self->mat->cols;
    PyObject *s, *a;
    if (PyArg_UnpackTuple(args, "args", 1, 2, &s, &a)) {
        if (s && a) {
            if (PyLong_Check(s) && PyLong_Check(a)) {
                int r = (int) PyLong_AsLong(s);
                int c = (int) PyLong_AsLong(a);
                if (r >= 0 || r < sr || c >= 0 || c < sc) {
                    double val = get(self->mat, r, c)
                    return PyFloat_FromDouble(val);
                }
            }
        }
    }
    PyErr_SetString(PyExc_TypeError, "Invalid arguments");
    return Py_None;
}
```

- We were asked to build a version of NumPy using a Python-C interface, speeding up matrix operations.
- We used **loop unrolling**, **SIMD instructions** and **OpenMP** to significantly increase the speed of operations.

Project 1

CS 161: Computer Security

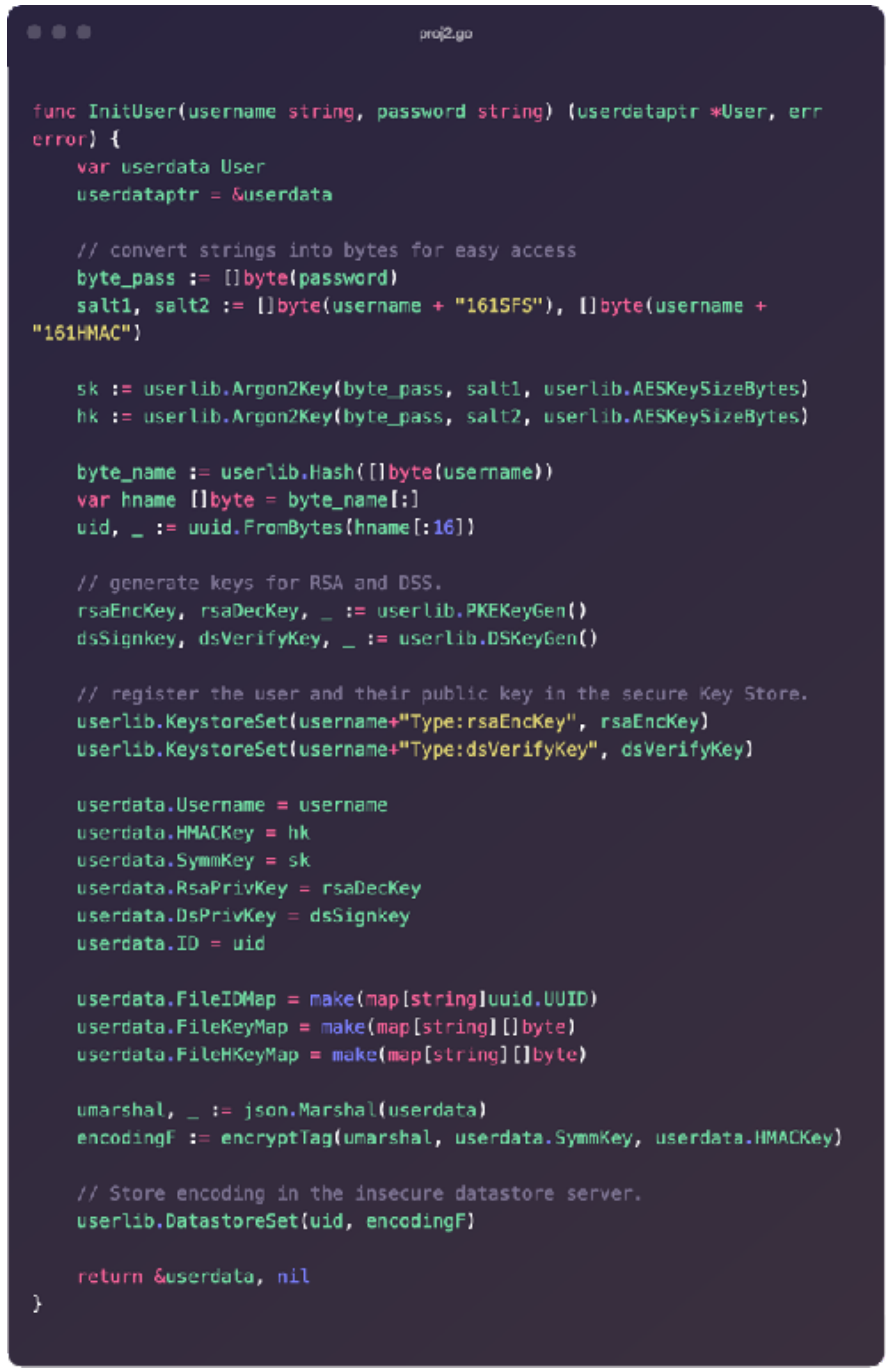


```
pwnable: $ ./exploit
dumb-shell $ cat README
[Epilogue.]

> run evanbot.exe < orbiter.dmg
Good work. You've found the blueprints for the Jupiter ships. Now the real
work can begin. We must not let the innovations of space exploration be
used for death and destruction. We can easily warn everyone on Earth and
the Moon, but communication with the people on Mars may be a challenge...

                *                           |
*                           .              \|/          .
                        *                 --*--            .
    .                                      /|\              .
         .                                / | \              .
         |                               /  |  \        .     .
  .____.----._                              |            .     .
_/            \___.--.                                    .     .
                       \____          *                    .     *
                             -.____            .            *
                                   \            *

[To be contunued in Project 2...]
```

- In this project, we were given a series of vulnerable programs on a virtual machine.
- Our goal was to use a series of increasingly complex memory safety vulnerabilities to access the secure data in these programs.
- We exploited vulnerabilities in the basic **C functions**, **ASLR**, **off-by-one errors**, and **stack canaries**.

Project 2

CS 161: Computer Security



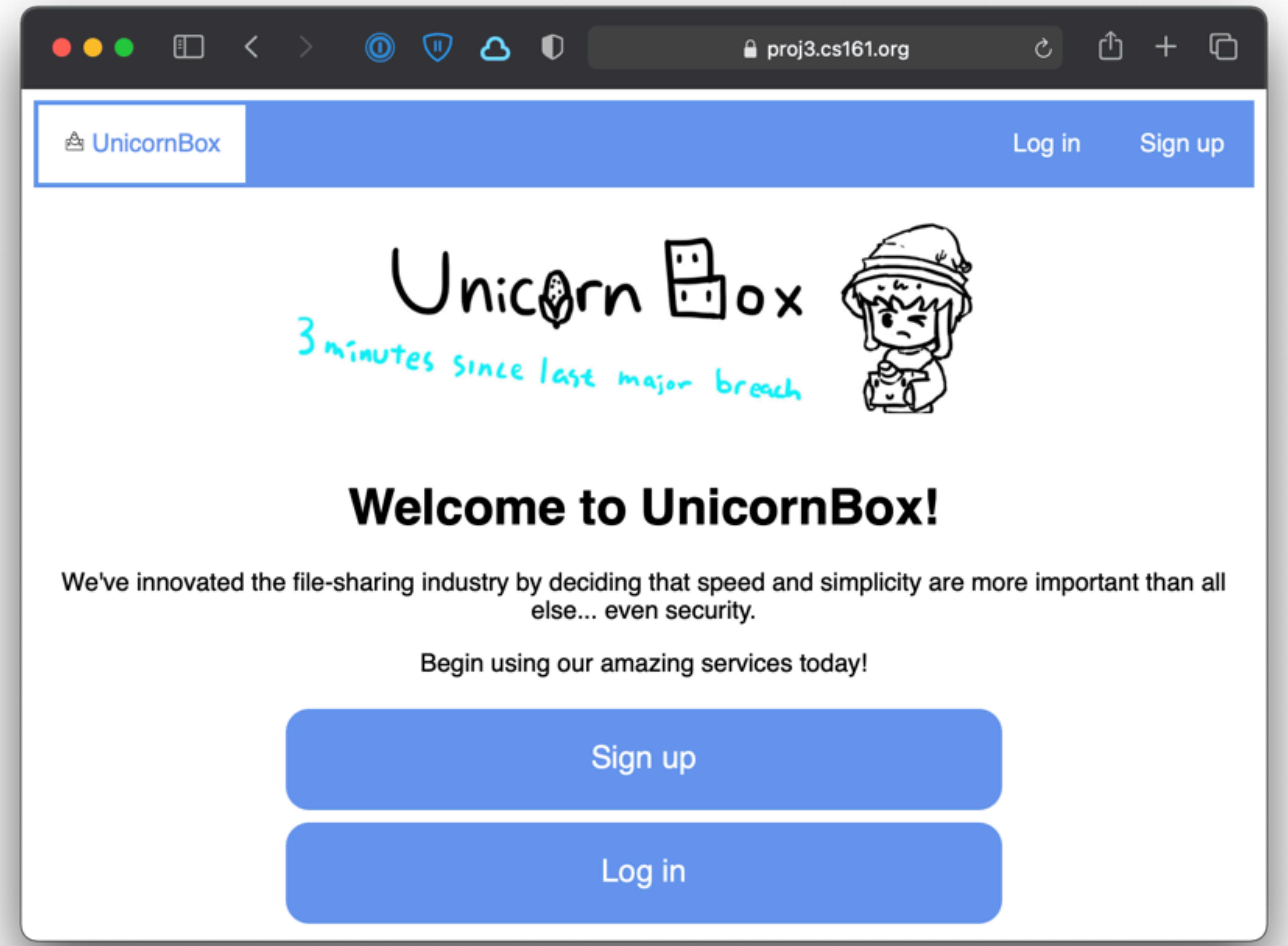
proj2.go

```go
func InitUser(username string, password string) (userdataptr *User, err
error) {
    var userdata User
    userdataptr = &userdata

    // convert strings into bytes for easy access
    byte_pass := []byte(password)
    salt1, salt2 := []byte(username + "161SFS"), []byte(username +
"161HMAC")

    sk := userlib.Argon2Key(byte_pass, salt1, userlib.AESKeySizeBytes)
    hk := userlib.Argon2Key(byte_pass, salt2, userlib.AESKeySizeBytes)

    byte_name := userlib.Hash([]byte(username))
    var hname []byte = byte_name[:]
    uid, _ := uuid.FromBytes(hname[:16])

    // generate keys for RSA and DSS.
    rsaEncKey, rsaDecKey, _ := userlib.PKEKeyGen()
    dsSignkey, dsVerifyKey, _ := userlib.DSKeyGen()

    // register the user and their public key in the secure Key Store.
    userlib.KeystoreSet(username+"Type:rsaEncKey", rsaEncKey)
    userlib.KeystoreSet(username+"Type:dsVerifyKey", dsVerifyKey)

    userdata.Username = username
    userdata.HMACKey = hk
    userdata.SymmKey = sk
    userdata.RsaPrivKey = rsaDecKey
    userdata.DsPrivKey = dsSignkey
    userdata.ID = uid

    userdata.FileIDMap = make(map[string]uuid.UUID)
    userdata.FileKeyMap = make(map[string][]byte)
    userdata.FileHKeyMap = make(map[string][]byte)

    umarshal, _ := json.Marshal(userdata)
    encodingF := encryptTag(umarshal, userdata.SymmKey, userdata.HMACKey)

    // Store encoding in the insecure datastore server.
    userlib.DatastoreSet(uid, encodingF)

    return &userdata, nil
}
```

- In this project, we were asked to apply the cryptographic primitives taught in class to design and implement an end-to-end, encrypted file sharing system.
- The client, written in Go, will allow users to login, save and load files from the server, overwrite and append to saved files, share saved files to other users, and revoke access to shared files.
- We applied our knowledge in **hashes**, **symmetric** and **asymmetric key encryption**, **public key agreement**, and **digital signatures**.

Project 3

CS 161: Computer Security



proj3.cs161.org

Welcome to UnicornBox!

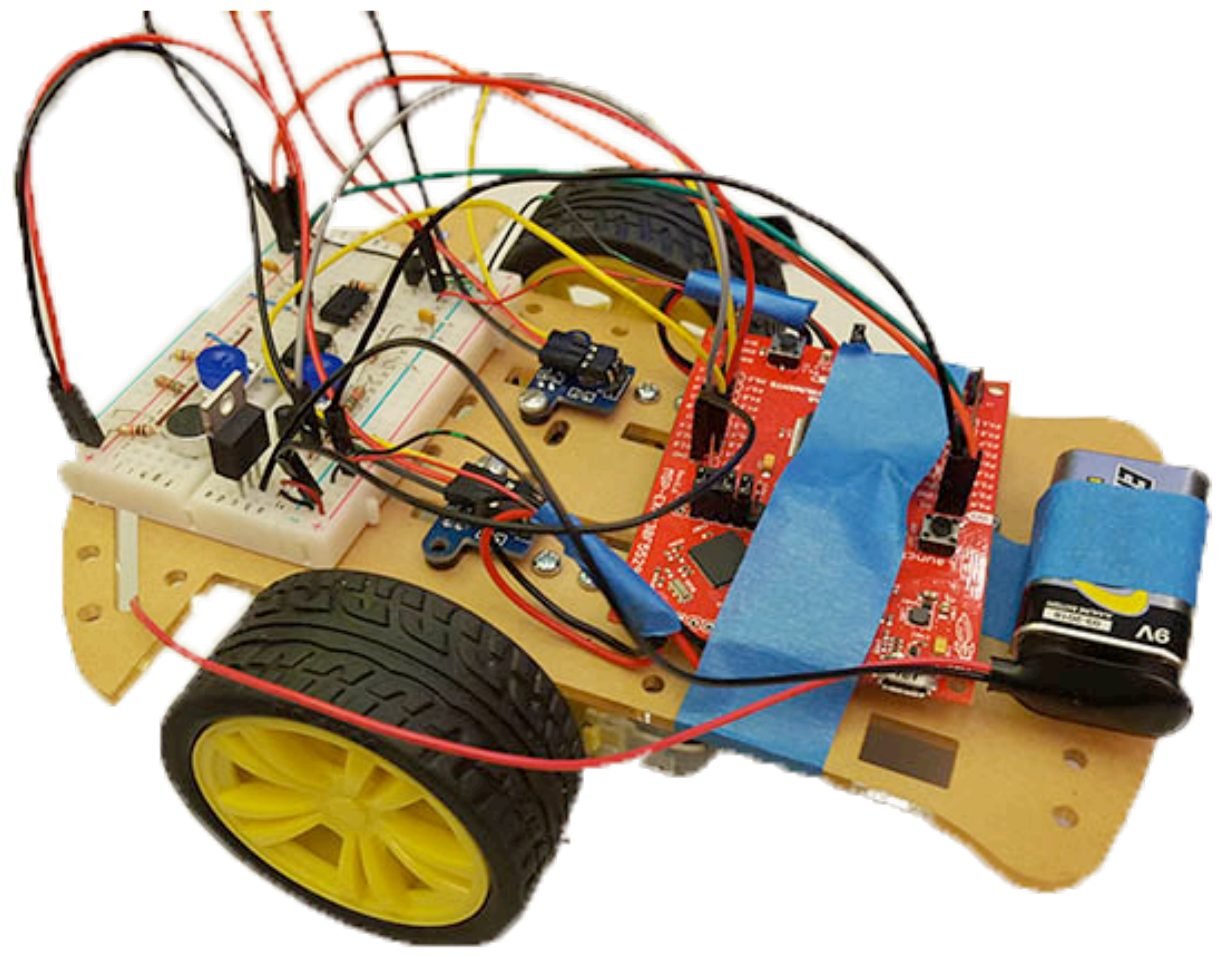
We've innovated the file-sharing industry by deciding that speed and simplicity are more important than all else... even security.

Begin using our amazing services today!

- In Project 3, we were tasked with using various web and network exploits to attack a staff-designed, poorly-secured web server and obtain “private” information, such as usernames and passwords.
- We performed exploits such as **SQL injection**, **stored XSS** and **reflected XSS** attacks.

SIXT33N

EECS 16B: Designing Information Devices and Systems II



- In Project 3, we were tasked with using various web and network exploits to attack a staff-designed, poorly-secured web server and obtain “private” information, such as usernames and passwords.
- We performed exploits such as **SQL injection**, **stored XSS** and **reflected XSS** attacks.