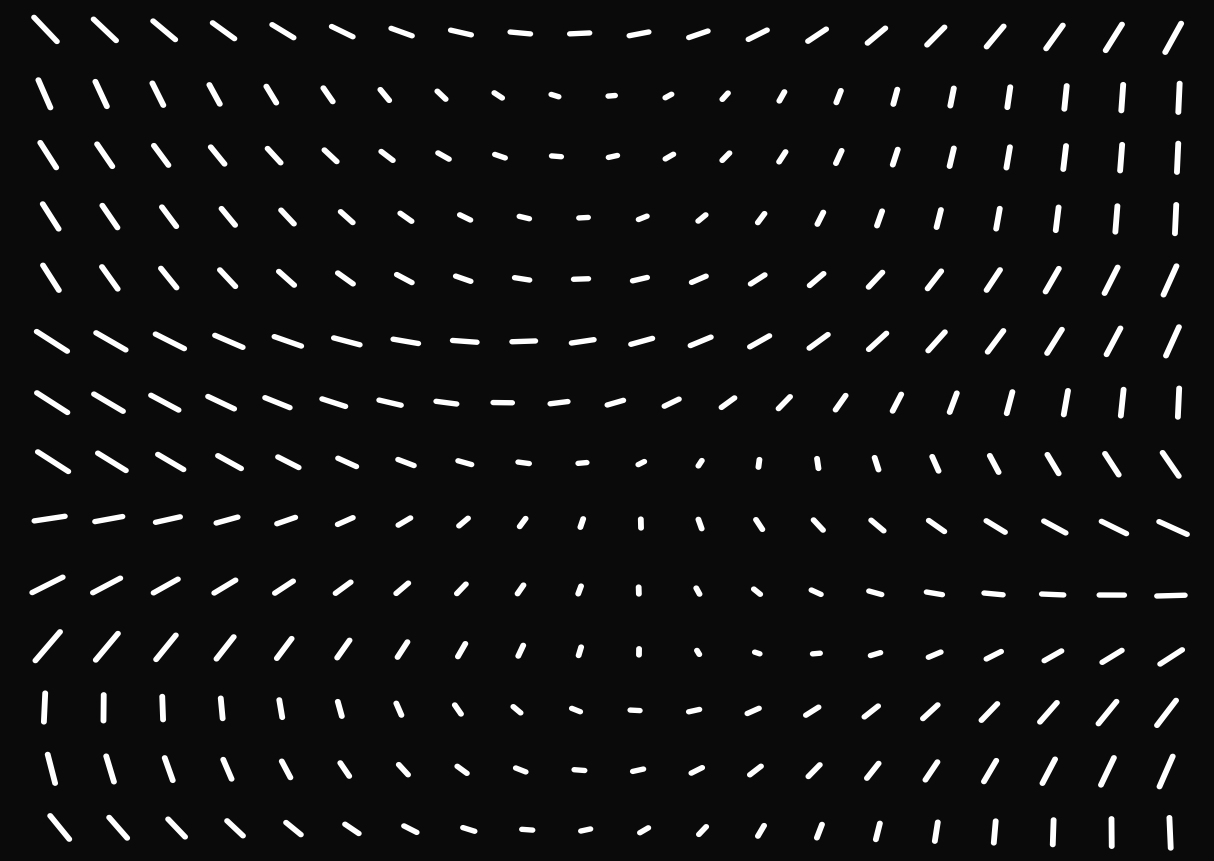


Simulations → Evolve over time

- ↳ Weather Forecasting
- ↳ Flight Simulation
- ↳ Physics problems
  - ↳ Solve with a computer

↳ Use math to model real-world phenomenon

Math to draw & see how things change

How to make a medium scale program

Serve as our medium sized application

- ↳ Advanced OOP
- ↳ Write sophisticated algorithms
- ↳ Testing & evaluating application

Syntax → Using Objects → Creating Objects → Understanding Inheritence & Polymorphism → Using libraries

C# is our primary language

C# in .Net | MonoGame | Visual Studio

C# like java
↳ runs in a framework

↳ Compiled, managed OOP
↳ Requires Common Language runtime
↳ Runs on multiple platform

We will be writing the libraries

How to succeed

Notebook → Phone → Enthusiasm

Exercises in class

Assignments are interconnected

Review → Data structures → Operator overloading → Managem → Test 1

↳ Delegates & Events → Logging → Recursion → Packages → Test 2
↳ Exception vs Assertions → Performance → Dependency injection → Final

Creating a Maze

Mazes → Primary Theme

Assignments create a 2D maze
Assignments work together

Works, good design, well tested, performant

1
Drawing a Maze
↓
2
Creating a maze game (Draw in more game)

3
Generate a maze w/ recursion
↓
4
Generate maze w/ Hunt kill

5
Performance, Dependency Injection
Simple AI

Cli

devenv

Install Tools

dotnet new console

dotnet new

dotnet sln add .

global.json

Start Solution

devenv ..... sln

Simulations

Aug 23rd 2023

C# & .NET

.NET supports a bunch of languages
- VB
- C#
- F#

.Net Framework like the JVM

Powerful IDE support

.NET specifies how projects are built → Needed Maven in java

Indicate what are the source files
Establish output
Manage Additional resources

C# Project file manages the resources

VS Code provides templates

How .NET manages execution

C# compiled to Intermediate Language (IL)

IL stored in an assembly (.exe or .dll)

Virtual Execution System = Common Language Runtime

- CLR performs JIT to conver IL to native machine instructions
- automatic garbage collection
- exception handling

C# slower startup but faster execution

Diagram Slide 7

Java

C#

Similarities

Diff

C# dll and exes
no default namespaces in C#
Name spaces in C# can be nested

Assemblies & namespaces are integral

Assemblies
↳ .exe .dll defined by csproj
↳ version # fixed

Namespaces
↳ Like java packages
↳ Group common functionalities
↳ Avoids naming conflicts

C# unified type system

All types inherit from Object

Can implicitly convert to Object

Value Types have all
System.Object methods
(tostring, equal)

Obj Two types

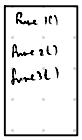
Value Types

Directly contain values
Copied by value
Can be turned into a nullable type
Local variable directly on the stack

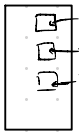
Reference Types
Contains references to objects
Can be null
Reference points to address in heap
Garbage collected when no longer referenced
Eg Class, interface

Track of all functions we call

Stack



Heap



Garbage Collection

Const

const fixes a value at compile time
program cannot change the value of a const

Read only

fixed value that is fixed at runtime

↳ like we want from a file, when we run it it can read from a config file then that cannot change

Demo

> dotnet new console -n NamespaceDemo
↳ in a folder named NamespaceDemo

adding reference vs using
↳ namespace within the dll
↳ references everything in the dll

We will be splitting into libraries

Properties replace getters & setters

C# properties are part of the language

Provide a way to get/set private fields

Separate from methods

Can have backing fields

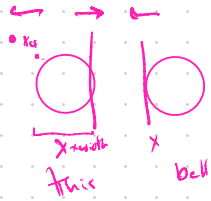
get must return the correct type

Properties Demo

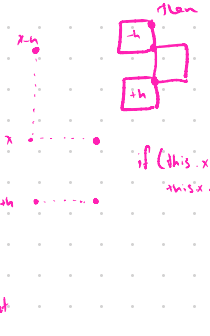
Add id

Properties are always public to be accessed outside

Private for fields
→ good for validation or internal only things

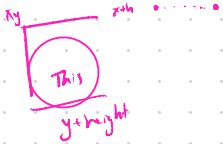


if current position edges touch side vector then flip the other



Check Collision

this.x + width == ball.x



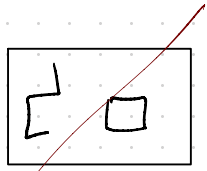
Similarly

Check height

Two Dimensional Mazes

Maze
is walls & paths

- Rectangular of odd size (9x7, 3x3, 3x5) odd nums
- Edge must all be walls
- Upper left corner always empty (1,1)
- All empty blocks must be connected
- X goes along the width y-height



(0,0) x (5,0)

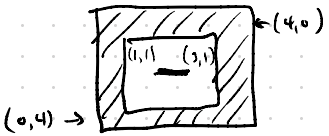
y

(0,5)

how monogame / windows does it. matches indexing on a screen

- X,y coord matches monogame

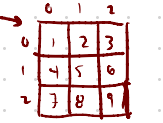
Off by 1 problems



Represented by a 2-d array of blocks

- Each position in the array is a wall or a path (empty block)
- Notice: indexing

```
var array = new int[3,3];
```



array[2,1]

array that stores solid or empty

Maze generators use directions

When generating,
↳ Describes a way a player can move

5 Dirs (cardinal) + None

Matrix of directions to convert into block maze

Bitwise encoding

- ↳ Common
- ↳ Multiple pieces of info
- ↳ Single value represent multiple
  - ↳ 1 num = E & W
- ↳ Value based Enum
- ↳ Values combined and keep orgin info

Binary

0	0000
1	0001
2	0010
4	0100
8	1000

C#

- ↳ Supports bitwise operators
  & , | return true or false based on bits
- ↳ Assign value for enum to each direction
- ↳ Can be combined w/ | and tested with &

```
Var dir = Direction.E;
dir = dir | Direction.S
enum Direction = {
```

```
0 0 1 0
1 0 1 0 0
0 1 1 0 (6)
```

OR

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

AND

A	B	A & B
0	0	0
1	0	0
0	1	0
1	1	1

```
Var isWest = (dir & Direction.W) > 0
Var dirs = new Direction[2,2];
```

	0	1
0	0110	0
1	0	0

Pitfall gotta do single | and &

```
enum Direction { None = 0, N = 1, E = 2, S = 4, W = 8};

Var dir = Direction.N;
Var otherDir = Direction.E;
Var combine = dir | otherDir; <- combining the two
Console.WriteLine(combine);

Var isEast = (combine & DirectionE) > 0;

Console.WriteLine(isEast);
```

Combine East

```
  0011
& 0010
------
  0010
```

$> 0 \rightarrow 2 > 0 \Rightarrow$ True

Create a block maze from a Direction maze

1) Determine the size of the block matrix w/ respect to dir matrix
2) Loop throug block Matrix and set everything to solid
3) Loop through columns then row
   - ↳ Empty block at (0,0)
   - ↳ <- Missing info
   - if east, add empty block
   - if south add empty block
4) repeat 3 to until the maze is generated

Direction Array in textfile

MapGrid Block[,]

	0	1	2	3	4
0	S	S	S	S	S
1	S	E	E	E	S
2	S	E	S	E	S
3	S	E	S	E	S
4	S	S	S	S	S

(5×5)

Direction [,]

	0	1
0	E|S ,	W|S
1	N ,	N

(2×2)

Euclidean vector

$$\| \sqrt{x^2+y^2} \|$$

Operator Overloading

Motivating Example

myName ==

- Java doesn't support operator overloading
- Needs to .equals to compare

C# overloads == for strings

Can define operators for their own behaviours

Other languages like C++ & Python can do this as well

Operator functions

is overload by declaring an operator function

Rules

name of a function
public & static
parameters is the operands
return type is what's returned

```
public class Fraction{
{
    private readonly int _numerator;
    private readonly int _denominator;
```

```
var a = new Fraction(1,2);
var b = new Fraction(1,2);
var c = a+b
```

```
public static Fraction operator + (
            Fraction a, Fraction b)
{
    int num, denom;
    num

return new Fraction(num, denom);
```

Can be overloaded:

+, -, *, /, %, ++, --

relational in pairs

== , !=
> , <
>= , <=

Must return a result

Casting Can also be controlled in C#

Explicit

We tell it to
using ()

int a = (int) 2.5;
→ downcast (loses information)

double b = (double) 2
→ upcasting (gain info)

Implicit casting

Invoked by compiler

Some types are upcasted by the compiler

double c = 5;

Int can be implicitly converted to a fraction

3 = 3/1

↪ automatic conversion to a wider type

```
public static implicit operator Fraction(
                                int a)
{
    return new Fraction(a, 1);
```

Explicit conversion should also be defined

We determine what information will be lost

```
public static explicit operator int (Fraction a)
{
    return a._numerator / a._denominator
```

IMap
IMap Vector
IPlayer

CreateMap()
↳ Now create the map

Object Browser

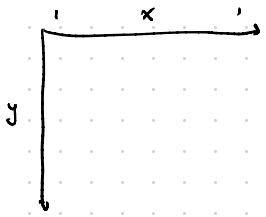
Settings → General → Advanced → Change Path → Org path under my name.

Turn directions into vectors

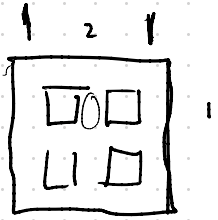
Map : IMap
↳ Knows Player
Goal
Map Grid

Create Map
↳ Lots of work
Creates player
Creates goal

IMap Provider
↳ Dir grids



Maze Library

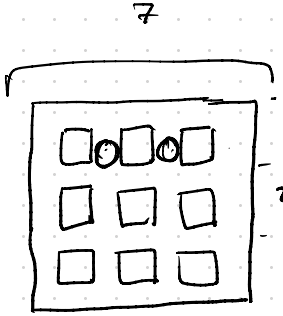


2+2

5 x 5

2+2

Count border_block



2 x 2

2+2

5 x 5

3 x 3

2 x 2

2 x 2

7 x 7

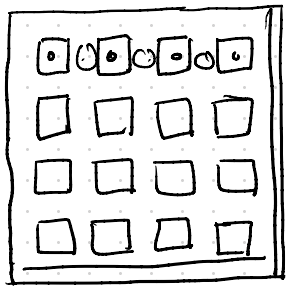
4 x 4

border_count

3 x 3

9 x 9

height is (DirMap.length * 2) - 1 + 2



DirMap Length - 1

4 + (4 - 1) + 2

3

For each Direction, we check what they contain then we add them together as vectors

Mocking & Unit Testing

Should feel familiar

Properties in C#
- for get or set
- can have backing field
- for users to use (other devs)

```
public int Count {get; set;}

public string Name {get;}

private double _time;

public double Time{
    get{ return _time;}
    set{ _time =value;}
}
```

Interface describe how the public interacts

- we can add private fields/methods for internal use

Expression body

- =>
- for simple statements

```
public int Length => 0  // only getter

public double Hours
{
    set => seconds /3600
    set => value * 3000
}
```

Unit testing

- Ensure reliability
- Test for bugs
- Separation of concerns
- Document how we expect code to work

MSTest

- Built in to .Net
- Templates built in
-

dotnet new mstest

Maze uses MazeTests

Arrange → Act → Assert

↳ Tricky bit

```
[Test Initialize]
public void TestSetup(){}
```

Mechanism for exception

```
[TestCleanup]
public void TestCleanup(){}
```

Single testcase can validate multiple pieces of data

Attributes on test cases

Can add new values easily

```
[DataTestMethod]
[DataRow(1)]
[DataRow(2)]
public void ValuesTest(int e){}
```

> runs twice with e = each

Assert classes make validation easy

Interfaces abstract implementation details

↳ other classes won't need to know specific details

↳ Creates a contract

↳ Can be mocked during unit testing

Test shouldn't need files on disk MockProvider

Moq lets us mock objects
↳ Don't always have all the objects
↳ Grade book needs IStudent
↳ Moq makes

```
var student = new Mock<IStudent>();
Student.setUp(s=>s.name).Returns("John Smith");
Student.setup(s=>s.LetterGrade).Returns("F");
```

Mock an IMapProvider

Simulations

Class Test 18th

Interfaces abstract implementation details

interfaces allow other classes to avoid understanding details of implementation
- Create a contract
- Can be mocked for unit testing
- Separate concerns

public class GradeBook {

public List<IStudent> Students {get; private set;}

Moq (library) lets us mock objects

Unit testing → don't always have the implementation
Use mocking to fake an obj with the expected results

```
var Student = new Mock<IStudent>();
Student.Setup(s => s.name).Returns("John")
```

Test map

Player needs a map?

Unit test w/

What if the unit test does not test everything

Useful?

Code exercised? How much of the code has been tested

Branches covered?

Code Coverage

Instruments our code

Leverage existing debugging

Still requires unit test or interaction to work

OpenCover (Windows only)

Requires Report Generator to visualize the results

filter cares about namespaces

only source code in git

Continuous Integration

is ensures tests are always run

Modern dev
↳ continuous integration or auto building

Gitlab supplies CI Tools

CI
↳ Build Code
↳ Run Tests
↳ Deploy

Build Check for compilation errors in debug & release
↳ Test - Run tests → Benchmarking
- generate error report
↳ Publish Create shippable release product
Save last version of the product

Debug vs Release

Debug allows us to use the debugger for Devs

Release has more optimizations

Give release to customers

.gitlab-ci.yml

↳ Describe steps for gitlab to run on our code

↳ Must be run on environment

Docker! woohoo :(

```
dotnet:
  image:
    mcr.microsoft.com/dotnet/sdk:6.0
```

Container has all tools we need

MS provides images

Dotnet SDK is required to compile & test dotnet apps

Docker containers are ephemeral
↳ Disappear after running
↳ Like stay for 5pm

Based on Debian

Stages & Jobs define a pipeline

Stage
- Build
- Test
- Deploy

Example on slides

↳ A job is defined by the developer describes what to do in a stage

→ Stage can have multiple jobs

→ Build has two jobs
↳ Build release and debug

gitlab puts us in the project root

Test next week Monday

.gitlab-ci.yml defines build process

- tells gitlab how to build & test the product
- what performs on pushes (by default)

Containers woohoo!

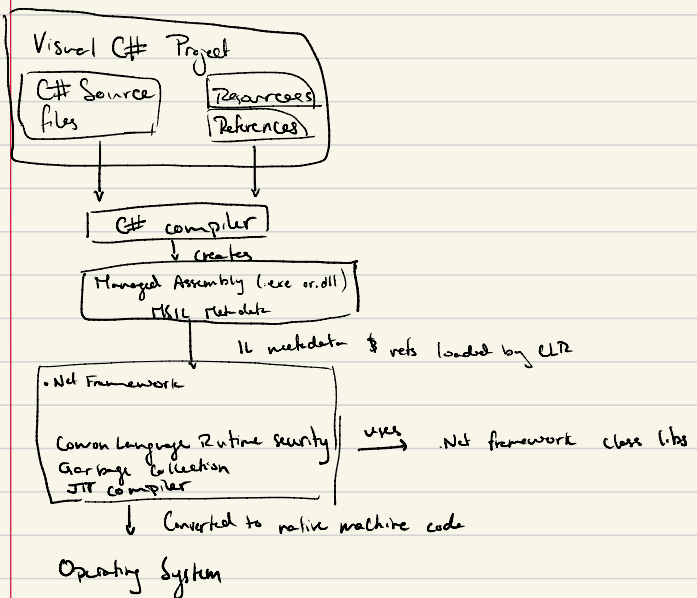
Stages & jobs define the pipeline

Checks the exit code, 0 for success anything else

Do CI minutes count Total of all jobs or pipeline time

Test failures from dotnet test return status code 1

C# Basics


Visual C# Project
C# Source files
Resources
References
C# compiler
creates
Managed Assembly (.exe or .dll)
MSIL Metadata
IL metadata & refs loaded by CLR
.Net Framework
Common Language Runtime Security
Garbage Collection
JIT compiler
uses
.Net framework class libs
Converted to native machine code
Operating System

Specifies how to build proj

.csproj manage aspects of the files

Sim java C++

OOp
Compiled
Multi platform
Garbage Collected

Df

.Net (C#) gens DLL & exe
no default namespace
namespaces can be nested

Bitwise encoding

Translate Single value can represent multiple binary options
Can be combined 0100 | 0010 = 0110
0111 & 0001 = 0110

```
public class Fraction{

    int Numerator { get; set;}
    int Denominator { get; set;}

    public Fraction(int num, int den){
        Numerator = num;
        Denominator = den;
    }

    public static Fraction +(Fraction f1, Fraction f2){

        int num = f1.Numerator + f2.Numerator;
        int den = f1.Denominator + f2.Denominator;
        return new Fraction(num, den);
    }

    public static Fraction +(Fraction f, int i){

        int num = f.Numerator + (i * f.denominator);
        return new Fraction (num, f.denominator);
    }

    public static implicit operator Fraction (int i) {

        return new Fraction(i,1);
    }

    public static explicit operator int (Fraction a){

        return a.numerator / a.denominator;
    }

    public static explicit operator Fraction(int i){

        return new Fraction( i ,1);
    }

    public static implicit operator double (Fraction f)

        return (double) f.Numerator / (double) f.denominator;
    }
```

Implicit vs Explicit overloading

```
$ dotnet new classlib -n InterfacesDemo

public interface IStudent{

    String Name {get;}
    string Program {get;}
    int Grade {get;}
    char LetterGrade {get;}
```

```
public Class GradeBook{
    public List<IStudents> Students {get; private set;}
    public String Report;

    Public GradeBook ( ){
        Students = new List<IStudents> ();
    }

    Public void GenerateReport(){

        // make String
    }
```

```
public class Student{

public String Name {get; private set;}
Public string Program {get; private set;}
public int Grade {get; private set;}
```

← Backing field

```
char letterGrade { get{
                    if (
```

```
public Student ( string name, string program, int grade){
    Name = name;
    Program = program;
    Grade = grade;
}
```

← no validation here

```
private int _grade;
public int Grade { get{
                    return _grade;
                   }
                   set
                   {
                    if (value < 0 || value > 100){
                        throw ——
                    }
                    _grade = value
                   }
                  };
```

```
dotnet new mstests -o GradeBookTests
dotnet add reference GradeBook.csproj

[TestClass]
Public class GradeBookTests{

    [TestMethod]
    public void GenerateStringTest ( ){
        // Arrange
        Student s = new Mock<IStudent>();
    s.setup    (s => s.name).Returns ("Jake"),
    s.Setup(s => s.grade).Returns (85)
        .Add (s.Object)

    // ACT

    //Assert
```

```
yml
image: mcr.microsoft.com/sdk:6.0

debug:
    dotnet build -c debug
build release
    Stage: build
    Script:
test:   - dotnet build -c release
    Stage: build
    Script:
        - dotnet test consoleapp.sln
```

Notes Backing fields

Code Coverage

Tool to see exercised

↳ Leverage debugging

- → Lines covered
- → Branches ... → hard to test all

→ What code is tested

100% → really impossible

Doesn't check the behaviour of the tests

Monogame?

Delegates & the Strategy Pattern

ASS2 Creating more in monogame.

NUget packages

mgwindowsgx -n <NameOfProject>

Derives from Game

program.cs is our main

Game.Run()
→ runs the game

graphics Device → sets up what will be shown

Initialize & LoadContent() → how to load our resources
→ each run once

Initialize → setup the initial state

Draw & Update → run in the loop

_SpriteBatch → responsible for drawing on screen

_SpriteBatch.Begin()
_SpriteBatch.End() → _spriteBatch.Draw(_box, Vector2.Zero, Color.White); → can add another sprite
→ Rectangle → to select parts of the image to show
→ where we draw the sprite

if (Keyboard.GetState().IsKeyDown(Keys.Right));

Keyboard input monogame

Assignment
- For Ass 1
- ScatterGL
- Run manager
- CI

Delegates & Strategy Pattern

Design Patterns — Language agnostic
- NVVM

Strategy Pattern
- Design classes to be flexible (& testable)

Delegates
- allows methods to be passed as a parameter
- Pass the strategy of how to do something as a param

Opt 1 Bool Param

```
public static void SortEmployees(Employee[] data, bool ascending){
```

- Allows algo behaviour to be picked at run time
- Implemented w/ delegate
- Could also be via interface (How we mocked)

Delegates
- Allows methods to be first-class
  - assigned to a delegate instance
- Type not a class
  - Defines a type of delegate
    - That represents a method
- Instantiating Delegate type
  - encapsulate instance w/ matching sig / return
  - Invoke through delegate instance

```
public delegate int PerformCalculation(int x, int y);

class Calculator {
    public static int Add(int a, int b){
        return a+b;
    }
}

// in main {
PerformCalculation calDel = Calculator.Add;
Console.WriteLine(calDel(2,3));
```

Passing to a method

Delegets are reference types
↳ Must perform a null check

```
if(comp != null){

}
```

null check operator ?

```
Comp?.Invoke(a,b)
```

? for null & nullable features

```
public delegate int PerformCalculation(int x, int y);

class Calculator{
    public static int Add(int a, int b){
        return a+b;
    }
    public static int Multiply(int a, int b){
        return a*b;
    }

Win main {
    PerformCalculation calDel = Calculator.Add;
    Console.WriteLine(calDel(2,3));

Static void Foo
```

Adding multiple methods to delegate instance

Delegate → type
instance → list of invocations
To add a method use + operator

Multi cast delegate
↳ like adding multiple event handlers to a button

```
public delegate void TransformDel(int x);
Class Program{

    S.v. Cube(x){   }
    S.v Square(x){   }
}

S.v. M{
    TransformDel delObj = Square;
    delObj += Cube;

    delObj(2); → Calls square then cube
                 ↳ order not always guaranteed
```

Lambda

Shorthand for writing a method

lambda expression in delegate
↳ Anon function

Expression Lambda:

```
( — ) => expression;
```

Statement Lambda:

```
( — ) =>{
        :
        ;
}
```

Any lambda can be conv to delegate type

Cannot remove += 'ed anon method

a list of non-void methods will run but only the last value is returned

Already tells us what types must be

```
PerformCalculation delCalc = (a,b) =>{
    if(a<b){ return b-a};
    else{ return a-b};

PerformCalculation delCalc = (a,b) => return a+b;
```

Common Delegates

Action delegate

Takes in nothing, returns nothing

```
Action act = () => Console.WriteLine("Hello");
```

- Tends to act on something
- good way to do multicasting
- sequential invoking

```
namespace DelegateDemo;

public delegate int PerformCalculation(int x, int y);

public class DelegatesDemo
{
    public void Calculate()
    {
        PerformCalculation calculator = (a, b) => { return a + b; };
        Console.WriteLine(calculator(1, 2));
```

Actions, Singletons & Logging

Recall Action Delegate

Pass method to other methods
Action ( ) => void
Built into C#
Compile w/ Lambdas

```
Action act = () => Console.WriteLine("Hello");
```

Actions want to add

```
internal class Square
{
    void Draw(){
        C.WL("Draw Square");
}

internal class Circle
{
    void Draw(){
        C.WL("Draw Circle");
}
```

```
Public class Shapes
{
    List<Action> _actions;
    public Shapes()
    {
        _actions = new List<Action>
    }
    public void AddDraw(Action a)
    {
        _actions.Add(a);
    }

    for(var action in _actions)
    {
        action?.Invoke();
    }
```

null check

Can now draw shapes w/o knowing details

Shapes provides way to add delegates
↳ Draw() calls all delegates
↳ Shapes doesn't need to know drawing details

Input manager
↳ Keys to code

Do we want multiple shapes to be possible

Singleton → Classic

↳ only build object once
↳ often will Lazy instantiation
↳ Constructor is private

```
(No inheritance)
Cannot be used as a base
public sealed class Singleton
{
    private static Singleton instance = null;

    private Singleton()
    {
    }
    public static Singleton Instance
    {
        get
        {
            if (instance = null){
                instance = new Singleton();
            }
        return instance;
        }
    var s = Singleton.Instance
```

Input manager
↳

Logging

Ensuring our code behaves properly.

Logging monitors program activity

- Structured way to record what a program is doing
- Logs can output to multiple places
  - Console
  - File
  - Database

NLog

- highly configurable
- Commonly used
- Relatively simple
- When shipping a product
  Provide nlog.config → XML file

Target

Defines where logs should go
- Files, Consoles, DB,

Rules

what gets logged

NLog

- Static method to fetch the logger
- logger class contains different methods for logging

Logging adds overhead

- comes at a cost
- More logs → slower code
- More info → Bigger log files
- Rules controls which log level
- Debug & trace only to be present in debug
- Release should have

NLog must live beside the folder

Copy to output Directory

Only care about the last run
Clear after restart

Recursion

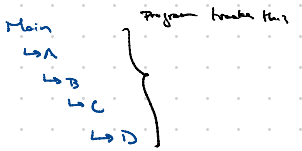
Obj
- use for A3
- more generation randomly

Ex
- → Showing folders / files tree

prob → Don't know how many layers

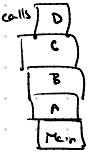
Recursion
- ↳ allows function to call itself
- ↳ usually have param to change functionality

for A calls itself creating a loop



- ↳ has limits
- ↳ repeated application of a recursive procedure or definition
- ↳ Consequences to calling yourself
- ↳ if we never exit → stackoverflow

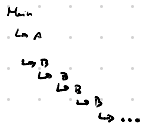
Call Stack
Tracks all current calls



Adding up to a given number can be done recursively

Sum(3) 0 +1 +2 +3 = 6

Recursive call stack



how to solve w/ & w/o recursion

```
Sum(n){
    int sum = n;
    if (value == 0)
    {
        return sum;
    }
        return Sum(value-1) + sum;
```

Sum(3)
↳ 3 + Sum(2)
↳ 2 + Sum(1)
↳ 1 + Sum(0)
↳ exit

Stack overflow
↳ no exit

```
public int Count
{
    get
    { return Count; }
    Set { Count = value; }
```

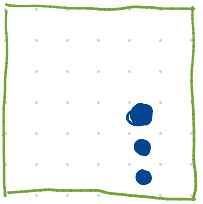
Search through folders recursively

Use recursion to loop through hierarchy of folders

- Print current folder name
- tabs to delineate
- behave similar to tree

loop dirs[]

printDirectories(dirs[i], level+1);



Internal Classes & Debug Assertions

Fun Design Stuff
Robustness ++

Access Modifiers

Java
Private → in the class
Public → anyone
Protected → any class in the package → Package (or package private) default

C#
Private
Public
Internal → Default (sort of) → Classes structs records
method/fields is private by default

Abstracting object creation protects Libraries

Returns an interface

↳ MazeFromFile implements IMazeProvider
↳ Avoids concrete class creation
↳ Allows implementations to be internal

↳ Done w/ Factory or Creator

↳ Create static method → returns IMazeProvider

Return Interface

↳ Return obj of any subtype
↳ EmployeeCreator

```
public Employee

internal class Employee IEmployee

public static class EmployeeCreator{
    public static IEmployee
        CreateEmployee(String type){ ← Enum
        if (partTime){
            return new PTEC();
        }else{
            return new EC();
```

Internal classes can be tested!

[assembly: InternalsVisibleTo("MoreRecursionTests")]

Assertions & Exceptions

Might not always want exceptions

Exc → Thrown to convey errors to caller
↳ Usually caused by external event

Expensive to use poorly
↳ Each exc = new obj
↳ Should not handle normal conditions
Try/Catch outside our control

Assertion
↳ ensures a statement about our code is true
↳ if not, the program throws an error & stops

Debug.Assert
↳ Built in
↳ Rendered out at release time
↳ Release does not keep Debug. stuff

↳ Asserts are good to validate private

↳ Start or end of internal/private

When Not
↳ Don't for public method validation
↳ Not to make sure the app works (no side effects)

Assertions
↳ Logical impossibilities

Debug.Assert(in [N, E, W, S])

Exc
↳ Exceptions are possible

```
public class Piano : IPiano {

    private List<(uint, string)> _things;
```

Dirks Changes

Check if provider is null
↳ ArgNullExc

Check if negative in CreateMap(x,y) >0
throw argRange exc

CreateMap()
Debug.Assert(_provider != null)
~~Log/Create = provider.CreateMap();~~

TransformToGrid
↳ Debug.Assert(map not null)
↳ Check width/height after creation

Internal

make Map Internal

public static class MapFactory{
public static Map CreateMap(IMapProvider provider){
return new Map(provider);

publish just mapGame

Can you cast — ll to an int

Clear up A3

Goal

Lib for mazes

Save Direction Map

↳ helps debugging

MTRecur : IMapProvider

Pick a default size

Testing Random Elements

Random info use a seed => nullable int seed def null

(int? seed = null)

No Nom, right size

Check oblong size maps 4x2 2x4 8x5

Menu

Text on screen

↳ Font Sprite & Font → into content folder

↳ Draw string ↳ Add in mgcb

Font String Color

_font = this.content.Load<SpriteFont>("Font");

in Vall Fonts for UI

Fnt

No text input Box

Recursion screen size

D.Num

Walk through array

D map 2x2 $\begin{bmatrix} S|E & W|S \\ N & N \end{bmatrix}$

$\begin{bmatrix} N0 & N0 \\ N1 & N1 \end{bmatrix}$

(1,0)→

$\begin{bmatrix} \bullet\rightarrow & \\ & \end{bmatrix}$

keep 1st of vectors

Recursion

Generating a maze recursively

1. Create an empty direction map of the appropriate size
   a. Recall that IMapProvider provides two CreateMap methods, one that takes in a width and height

[DN, DE, DS, DW]

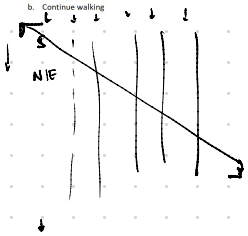
Random from this

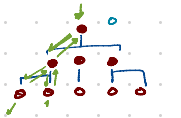
Select Random from enum

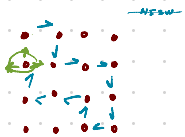
2. Pick a random starting vector (MapVector) S,E
3. Start walking along that vector
4. Once you have stopped walking return the map

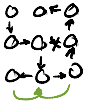
Walking

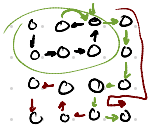
1. Walk along the provided vector
2. Keep a list of all the vectors you have previously visited
3. Loop through all the possible directions in a random order
4. Based on the chosen direction,
   a. Note the next forward position
   b. Note the opposite direction
5. If the next position is inside the map and that spot has not been visited
   a. Update that position in the map to contain both the forward and opposite direction
   b. Continue walking











Managing & Publishing Packages in C#

How to get in the hands of users

Managing Dependencies IS critical
- ↳ Logging
- ↳ Entity
- ↳ Mongoose
- ↳ JDBC

Build projects so they include our dependencies
How to publish & share our projects

dotnet integrates package management NuGet

Python does not, it uses pip to install

NuGet can be installed standalone

NuGet packages are just zips
- nupkg
  - ↳ contains → binary files
    - ↳ manifest file of meta data (version)
  - ↳ Nuget handles unzip & integration

Packages do not go into VCS (git)

Source Control stores only our source code
Build server → wherever we build our code
restore → fetches libraries
.csproj tells what to restore

Dependencies that rely on each other to make a dependency chain

.nuspec - describes the manifest of the package
- ↳ contains section to describe dependency chains
  - ↳ NuGet builds dependency graph
    - ↳ NuGet gets all required libraries

Package can contain more than one target
- ↳ Package can contain builds for multiple architectures & net versions
- ↳ Lib folder contains multiple versions
  - ↳ sometimes we can have a single lib

dotnet CLI wraps existing nuget commands
- ↳ dotnet add package
- ↳ dotnet restore wraps nuget & fetches
- ↳ can be accessed via dotnet nuget

We have to worry about licensing

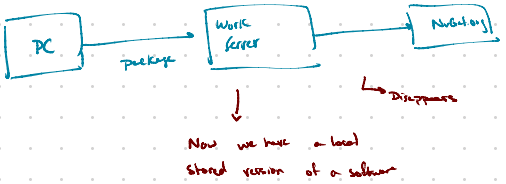
TIL

Overloads
Singletons
Logging
Delegates
Factory Pattern
Recursion

Packages can come from diff sources

online repo of NuGet open source

Packages can come from any service



Creating Packages is straight forward

↳ $ dotnet pack → makes the package
↳ Need to consider the architecture & compilation mode
↳ Specify Release or Debug

Make sure our package only contains release or debug information

Where does package auth / name come from?
↳ Copy file

pack does not strip an exe
↳ meant for libraries to develop with

Modern dotnet separates exe and dll

To reference local packages

$ dotnet nuget add source

by default, only looks online

Test Review

Operator overloads
↳ object implements it's own +, -, *, /, ==

Eg MapVector a,b;
c = a + b;

↙ Drill this

public static MapVector operator+(a , b){

return new MapVector(a.x+b.x, a.y+b.y);

}

Comparisons need pair overloading: (==, !=), (>,<), (>=, <=)
↳ Because it returns true or false we need both

Delegates
↳ lets us
↳ Treat methods as first class objects
→ Pass methods around
↳ Regular named method or w an anonymous method ()=>{}
→ Built in delegates Action ()=>{} void Action(void)

Delegate types
→ Cannot be defined in a class but inside a namespace

namespace Test{
public delegate double Compute(int a, int b);
↑ Type

class Test{

Compute computer;
void Add(Compute a){
double? c = a?.invoke(2,3);
}

public test (Action a){

a?.invoke();

Test question

Short programming (4)

- Overload == (explicit cast)
- Create a factory
- Create a delegate type
- Multicast / invoke delegates
- Create a singleton

Add((int a, int b) => {
↳ anonymous method
return (double)(a/b);
}

Strategy Pattern

↳ lets us define how a method can work later

Ask about naming

Dirk

↳ lets us decouple behaviour

↳ Determine behaviour at run time

Other Stuff

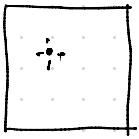
Internal Classes

↳ Singletons

↳ Factory Pattern

↳ Interface using a factory

Hunt kill



Exam Studying

Singleton

```
sealed class Singleton {

    private static Singleton _instance = null;
    public static Instance {
        get {
            if (!_instance){
                _instance = new Singleton();
            }
            return _instance;
        }
    }

    private Singleton()
    {
    }
}
```

Delegates

```
public Delegate char LetterGrade(int grade);

LetterGrade translator = (grade)=>{

    if(g<60)
    {
      return 'F';
    }
    else if(g>60 && g<70)
    {
       return 'D';
    }
```

translator += (grade) => { return 'F'}; → calls the last one in

Delegate objects can be null

check if they are null before invoking

```
    delegate?.Invoke(a,b);
}
```

implicit for upcast explicit for downcasts

```
public static explicit operator int(Fraction f){
    return f.num / f.denom;
}
```

Operator Overloading

```
public static MapVector operator+(MapVector a, MapVector b)
{
    return new MapVector(a.x+b.x, a.y+b.y);
}

public static bool operator ==(MapVector a, MapVector b)
{
    return (a.x==b.x && a.y==b.y);
}
```

Factory Pattern

used for internal classes

```
public class AppCreator
{
    public static IMapProvider AppCreator(string appType)
    {
        if(appType == "coolApp"){
            return new CoolApp();
        }
        return new BadApp();
    }
}
```

casting

Create an app that allows handwritten typing.

Operator Overloading

```
public static int operator+(int a, int b)
{
    return a+b;
}

public static int operator-(int a, int b)
{
    return a-b;
}
```

Delegate

```
public Delegate int Computer(int a, int b);

Computer add = (a,b) => { return a+b; }

add += (a,b) => { return a-b; }
```

Singleton

```
public static class Singleton{
    private Singleton _singleton = null;
    public Singleton Singleton
    {
        get
        {
            if(_singleton == null)
            {
                _singleton = new Singleton();
            }
            return _singleton;
        }
    }
```

Mock: IPlayer

next semester

→ She

```
public static implicit operator int(Factory f){
```

Factory

```
public static class Factory{
    public static IFactory FactoryCreator(string type)
    {
        if(type == "CoolFactory")
        {
            return new CoolFactory();
        }
        return new LameFactory();
    }
```

Create a method to add delegates

```
[TestMethod]
[DataRow( )]
public void Test( )
{
    //Arrange

    //Act

    //Assert
}
```

IMapProvider mockProvider = new Mock<IMapProvider>();

mockProvider.Setup(s => s.CreateMap()).Returns(_directions);

Mocks

Hunt Kill Maze Generation

Generate Maze -> Recursion

Easier than recursion

New menu item for Hunt & Kill

Specify size of the maze

Library implements IMapProvider
↳ Still need to test the algorithm

Implement both CreateMaps

1) Create empty Dir[,]

2) Pick random start position

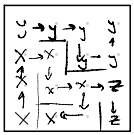
Walking

New position

if invalid return (-1,-1)

Hunt

Iteratively search for an unvisited position



← hunt time

hunting J → not touched, adjacent valid position

Once hunt fails the map is valid
↳ return dir[,] map

Hunt loop & Search loop

Performance Analysis

Concerns
- Time
- Memory
- Worst Case Scenario

Big O Notation

- Looks for worst case
- Approximate what happens if an algo goes to infinity

O_N → Linear
O_{N^2} → exponential

Analyze when algo goes to infinity

Look at largest op
- loops are often long

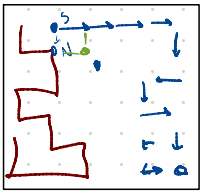
Two major operations
- Read to a val or var → Cheap
- Write to a val or var → expensive

Nesting loops Compounds speed problems

Stock Desktop
4gb ram to run Electron app
Loads all data into RAM

Check pos first hunted

hunt



Analyzing code performance

Stop Watch is a limited tool.

Limited:
- ↳ Precision
- ↳ Requires Code Modification
- ↳ Only monitors CPU usage

What about memory usage?
- ↳ How much space is it using
  - ↳ how does that change

Visual Studio has a performance profiler
- ↳ Analyze CPU
- ↳ Memory Usage
- ↳ Events, GPU usage, and other elements

Hot Path
where the most time is being spent
↳ can give red herrings

Performance profiler

Memory
↳ need to take snapshots

inclusive size (size of it + things in it)

Strategies for improving

Caching in loops

Reuse variables → re-using a random

Examine Overhead

Parallelizing loops is a standard for speeding up code

- if performing two independent

Parallel.For & Parallel.ForEach

Parallel.ForEach

Parallelizing the map generation

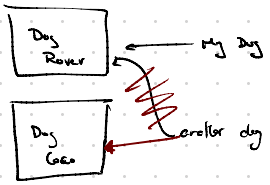
Passing By Reference and Out

Language Features

↳ Low level object play

Java always passed by value → copy of reference to an object
Passing the value of a reference address

HEAP



What happens when passing by reference?

Any ↳ ref → pass by val but it's the reference

Java always by value

C# defaults to pass by value

Copy of reference

ref Keyword
↳ passes the ACTUAL reference to the object

PbV
↳ copy
↳ Don't affect the original call

PBR
↳ Modify directly

ref

Out
↳ vars we don't create ahead of time

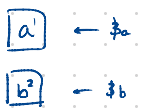
Ref Syntax

must be marked w/ ref

```
Void SquareRef(ref int x){ ... }
int initialized = 5;
SquareRef(ref initialized);
```

Pointer

Var who points to an object in memory




```
Void Swap(int a, int b)
{
    // Does not change in main
}
```

```
void Swap(ref int a, ref int b)
{

}
```

Swap($a, $b)

Out

Code the object in a method

```
ChangeName(out myDog);

ChangeName(out anotherDog){

}
```

like ref but it cannot be preinitialized

```
Dog myDog;
SetDog(out myDog);
Console.WriteLine(myDog.Name)
```

```
int a = int.Parse("3"); → can throw an exception
try{
    int a = int.Parse(num);   ← expensive!
} catch {

int a;
bool b = int.TryParse(num, out a);

if( int.TryParse(num, out a)
{
    a = a+2;
}
```

Change map to be a field in program

When benchmarking

- How many points?
- How many numbers?

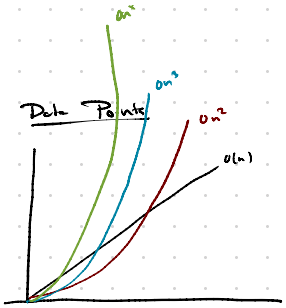
We Design our benchmarker

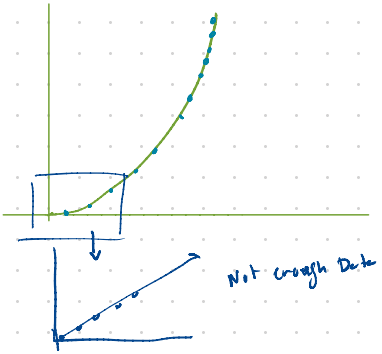
Improvement

- One algorithm
- Could be no usage

New Ones

- Hurt Kill Recursion Performance?





Cumulative Final Exam

Review

- C# Review
  - Properties
  - Release vs Debug
  - Unit Testing
  - Unified Type System →

Allows — Since everything is an object

Operator Overloads

- Operators (+, -, /, *)
- Casting
  - Implicit → Upcasting   double a = 1;   → no info lost
  - Explicit → Downcasting   int a = (int) 2.3;   → some info lost

Code Coverage

- Measure what code is used
  - Line Coverage
  - Branch Coverage
- Helps check all code is tested
- Does not check test validity

Delegates

- Make a method a first class object
- Passing methods as parameters or variables

1) create Delegate type
2) Create Delegate instance

```
delegate void MyDel(int a, double b);

MyDel a = (a, b) => {   }
```

(Enforces the types)

Lambdas

Lambda (anon funcs)

- Define delegates dynamically
- No types on parameters

Closure

Can access all variables in scope

```
int a = 1;
doble b = 2;
Action c = () => {
```

Input Manager

- Series of Delegates
  - Actions
- Match Keys w/ Actions
- Multicast delegate

```
a = () => {}
a += () => {}
```

Logging

- App should log to record info
- NLog
- Configurations
- Log levels
  - Debug
  - Info
  - Warn
  - etc...
- Log outputs
  - To console
  - To file
  - etc
- Recreate user action
- Monitor our application
- Performance

Encapsulation & internal classes

- What do users get?
- Only return an interface
- Libraries implemented w/ internal classes
  - only visible within the assembly
- User only cares that they return an IMapProvider.
- Can expose to tests

Factory Pattern

- with internal classes
- Create objects and return the interface

```
public static IMapProvider GetProvider(){
    return new RecursiveAlgorithm();
}
```

Recursion

- → Function calls itself
- → Stop condition
- → Starting point
- → Stack & overflows

Eg print directory tree using recursion

Patterns

Singleton

- One instance of an object
- Lazy instantiation
- Public & Private constructors

```
private _instance = null;
private constructor(){}
static get {
    if (instance == null)
    {
        _instance = Singleton;
    }
    return instance;
}
```

Strategy

- Providing a strategy
- Ex searching different ways
- Implement in input manager
- Interfaces or Delegates

Map(RecursiveAlgo B) ← Title in an IMapProvider

Performance

- Timing of a program
  - Execution time
- Memory
  (Heap vs Stack)
- Stop watches
- Big O notation
  - Terms (On, On^2)
- Experiments

Ref & Out

- Pass by reference vs value
- Stack vs Heap
- Why for performance reasons.
- Java & C#

Parallelization

- Parallel.For
- Overhead associated w/ parallelization

Performance Profiler

- Loops being expensive
- news Memory Expensive

Benchmarking
- Customer needs to know how performant I'm