MSO assignment 3, Abracha Koens (9995153) and Aron Hopman (0296163)

# **Part 1 Extending and adapting the software design**

Mermaid code in blue:

  classDiagram

    %%region UI

        namespace namesapce\_GenericUI{

            class UI1{

                %% We'll probably have to split this class up for higher cohesion.

                +void ClickRun()

                +void ClickMetrics()

                -string ReadTextBoxProgram()

                ...()

            }

            class IDataBridge{

                +void SetTextBoxProgram(string text)

                +void SetTextBoxOutput(string text)

                +string ReadTextBoxProgram()

                ...()

            }

        }

        UI1 \*-- ProgramImporter : 1 programImporter

        UI1 \*-- ExamplePrograms : 1 examplePrograms

        IDataBridge <-- UI1 : 1 dataBridge

    %%

    %%region Forms

        namespace namespace\_MSO\_P3\_Forms{

            class Form1{

            }

            class Form1.DataBridge

        }

        <<partial>> Form1

        Form1 \*-- UI1

        Form1.DataBridge \*-- Form1

        Form1.DataBridge ..|> IDataBridge

    %%

    %%region Application

        namespace namespace\_Applic{

            class Application{

                +void Run()

                #InnerProgram AskForProgram()

                #void UseProgram(InnerProgram program)

                -void Execute(InnerProgram program)

                -void GetMetrics(InnerProgram program)

            }

            class ProgramImporter{

                -string path

                -string importFromtxt(string fileName)

                -bool TryFindPath(string fileName, out StreamReader output)

                +InnerProgram ParseProgram(string fileName)

                -Body.Builder ParseCommandBody(string[] code)

            }

            class ExamplePrograms{

                +InnerProgram basic1

                +InnerProgram basic2

                +InnerProgram advanced1

                +InnerProgram advanced2

                +InnerProgram expert1

                +InnerProgram expert2

                ...

            }

        }

        Application \*-- ProgramImporter : 1 programImporter

        ProgramImporter ..> InnerProgram : creates

        Application \*-- ExamplePrograms : 1 examplePrograms

        ExamplePrograms --> InnerProgram

        ProgramMetrics <.. Application : uses

    %%

    ProgramImporter ..> Body.Builder : creates

    ExamplePrograms ..> Body.Builder : creates

    %%region Commands

        namespace namesapce\_Commands{

            class ProgramMetrics{

                +int commandCount

                +int maxNestingLevel

                +int repeatCommandCount

            }

            class Body.Builder{

                -Body.Builder AddCommand(ICommand command)

                +Body.Builder turn(Dir2 dir)

                +Body.Builder move(int stepCount)

                +Body.Builder repeat(int count, Body.Builder body)

                +Body.Builder body(Body.Builder addedBody)

                +Body Build()

            }

            class ICommand{

                +void ApplyOnWorld(ref ActualWorld world)

                +ProgramMetrics GetMetrics()

            }

            class Body

            class Bodied

            class Repeat{

                -int count

            }

            class RepeatUntil

            class If

            class ICondition{

                +Check(ActualWorld world)

            }

            class Turn{

                -Dir2 dir

            }

            class Move{

                -int stepCount

            }

        }

        <<interface>> ICommand

        <<abstract>> Bodied

        ProgramMetrics <.. ICommand : creates

        RepeatUntil --> ICondition

        If --> ICondition

        Body <-- Bodied : 1 body

        Body.Builder ..> Body : creates

        Body.Builder --> ICommand : \* commands

        Body.Builder ..> Repeat : creates

        Body.Builder ..> Turn : creates

        Body.Builder ..> Move : creates

        Body.Builder ..> RepeatUntil : creates

        Body.Builder ..> If : creates

        Body ..|> ICommand

        Body --> ICommand : \* commands

        ICommand <|.. Turn

        ICommand <|.. Move

        Repeat ..|> Bodied

        RepeatUntil ..|> Bodied

        If ..|> Bodied

        Bodied ..|> ICommand

    %%

    ICommand ..> ActualWorld : affects

    class InnerProgram{

        +WorldState Execute()

        +ProgramMetrics GetMetrics()

    }

    InnerProgram --> Body : 1 commands

    InnerProgram --> ActualWorld : 1 startWorld

    %%ProgramMetrics <.. InnerProgram

    %%region World

        namespace namespace\_World{

            class ActualWorld{

                +ActualWorld CopyState()

                +void TurnLeft()

                +void TurnRight()

                +void MoveForward(int dist)

            }

            class WorldSettings{

                %% Data that can't change while the Innerprogram is running

                -WorldCell[,] worldGrid

                +WorldCell GetCell(int2 pos)

            }

            class BlockException

            class LeftGridException

            class WorldCell{

                Empty

                Blocked

            }

            class WorldState{

                %% Data that can be changed by the program

                +WorldState Copy()

                +void TurnLeft()

                +void TurnRight()

                +void MoveForward(int dist)

                +void AddToTrace(IEventTrace event)

            }

            class PlayerState{

                +int2 pos

                +Dir4 dir

                +PlayerState Copy()

                +void TurnLeft()

                +void TurnRight()

                +void MoveForward(int dist)

            }

            class IEventTrace{

                +string TextualTrace()

            }

            class TurnTrace{

                -Dir2 dir

            }

            class MoveTrace{

                -int stepCount

            }

        }

        <<interface>> IEventTrace

        <<enumeration>> WorldCell

        ActualWorld ..> BlockException

        ActualWorld ..> LeftGridException

        ActualWorld \*-- WorldSettings

        WorldSettings --> WorldCell

        ActualWorld --> WorldState : 1 state

        WorldState \*-- PlayerState : 1 player

        WorldState \*-- IEventTrace : \* trace

        WorldState ..> IEventTrace : creates

        IEventTrace <|.. TurnTrace

        IEventTrace <|.. MoveTrace

    %%

    InnerProgram ..> WorldState : creates

    %%region Geometry2D

        namespace namespace\_Geometry2D{

            class Dir4{

                +Dir4 North $

                +Dir4 East $

                +Dir4 South $

                +Dir4 West $

                +int2 ToVector()\*

                +Dir4 Rotated(Dir2 dir)\*

                +void Rotate(ref Dir4 subj, Dir2 dir) $

                +int2 MovePoint(int2 point, int dist)

                +T Match<T>(T caseNorth, T caseEast, T caseSouth, T caseWest)

            }

        }

        PlayerState <-- Dir4

    %%

Design patterns:

* The use of IDataBridge looks like a design pattern to me, but I don’t yet know which one.
* …

Deviations from practical 2 design:

* Extracted abstract class ‘Bodied’ from the repeat-command, because there are now multiple command types that use a body in a similar way.
* …

# **Part 2 Implementation and code quality**

Measures taken:

* …

Refactoring examples:

* …

Changes inspired by metrics:

* …

# **Part 3 Evaluation**

Likely future changes:

* …

High cohesion:

* …

Low coupling:

* …

# **Part 4 Testing**

Test fails + reasons:

* …

# **Part 5 Work distribution & retrospective**

Task distribution:

Part 1: …

Part 2: …

Part 3: ...

Part 4: …

What went well:

…

What could have been better:

…

What we learned:

…