University of South Australia

UniSA STEM

**Object-Oriented Programming**

# Assignment
# Tons-o-Dice
# Part 1- Design

# Introduction

The assignment is intended to provide you with the opportunity to put into practice what you have learnt in the course by applying your object-oriented programming knowledge and skills. It will require you to apply object-oriented methods to design, implement, and test a text-based application software where users can play dice games.

The assignment is worth for 30% of the assessment in this course, and consists of two parts:

- Part 1 – Design (10%)
- Part 2 – Implementation (20%)

This document specifies the tasks for Part 1 of the assignment, UML design of the software, while Part 2 is be specified in a separate document, released at a later date.

This assignment is an **individual task** that will require an **individual submission**. You are required to **submit your work via the course website as prescribed in the Submission Details section.**

This document is a kind of specification of the required end product that will be generated by implementing the assignment. Like many specifications, it is written in English and hence will contain some imperfectly specified parts. **Please make sure you seek clarification** if you are not clear on any aspect of this assignment.

# Course Objectives

By completing this assignment, you are expected to partially fulfill the following course objects:

- CO1. Convert a problem statement into an object oriented solution
- CO2. Use inheritance and polymorphism to solve problems
- CO6. Analyse and explain the behaviour of object oriented programs

# Assignment Learning Outcomes

This assignment is for assessing the following learning outcomes:

- Design a UML class diagram using UMLet
- Apply object-oriented principles including encapsulation, abstraction, inheritance, and polymorphism to the software design

## The Assignment Task Specification

The task for the Part 1 of the assignment is to produce a UML class diagram of an application software, called the Tons-o-Dice, a text-based application software where users can play dice games.

As a text-based application software, Tons-o-Dice will use a command line interface where users type in a command or input and the relevant output is printed onto the screen in text. When the software is executed, it will show a greeting message and a list of commands, followed by a command prompt '>' indicating the system is ready to take user's input. At the prompt, users can give one of the following commands:

- n: new player, register an account for a new player
- c: show the number of coins this player has
- s: show the leader board, which will print the list of the players and their scores
- p: play a game, which will start a new game
- q: quit, which will quit the application

For registering a new player, the system will ask for the name of the new player. The name is not allowed to be changed after being registered, and also the name must be unique (i.e. not allowed to have the same name as an existing player). A new player is given 100 coins for playing games.

Below is an example of the output on the screen (user's input in ***bold italic***).

```
Welcome to Tons-o-Dice!
Developed by Alan Turing
COMP 1048 Object-Oriented Programming

What would you like to do?
 (n) register a new player
 (c) show your coins
 (s) show the leader board
 (p) play a game
 (q) quit
> n
What is the name of the new player?
> Alan
Welcome, Alan!

What would you like to do?
 (n) register a new player
 (c) show your coins
 (s) show the leader board
 (p) play a game
 (q) quit
> n
What is the name of the new player?
> Alan
Sorry, the name is already taken.

What would you like to do?
 (n) register a new player
 (c) show your coins
 (s) show the leader board
 (p) play a game
 (q) quit
> n
What is the name of the new player?
```

```
> Steve
Welcome, Steve!
```

Users can play dice games. Only one game will be played at a time, i.e., the game must be finished before playing another game. There are three dice games users can choose from: Even-or-Odd, Minz, and Bunco. Different dice games have different number of players required. Even-or-Odd is played by one player, Minz can be played by 3 to 5 players, and Bunco can be played by 2 to 4 players. For playing Minz or Bunco, users will be asked how many players will play the game at the beginning. If the number of players registered is not enough to play the selected game, an error message will be displayed and return to the main menu. Once the number of players to play the game has been decided, each player needs to input the name and how much coins to bid. If the name does not match to a valid registered player or if the player with the name is already in the game, an error message will be shown and asked again. Each player can only bid up to the number of coins they have. Depending on the game results, the winner will receive the amount s/he bid, while others will lose their bid. If the player has no coins left, then that player cannot play anymore.

```
What would you like to do?
 (n) register a new player
 (c) show your coins
 (s) show the leader board
 (p) play a game
 (q) quit
> p
Which game would you like to play?
 (e) Even-or-Odd
 (m) Minz
 (b) Bunco
> m
Not enough players to play Minz.

What would you like to do?
 (n) register a new player
 (c) show your coins
 (s) show the leader board
 (p) play a game
 (q) quit
> p
Which game would you like to play?
 (e) Even-or-Odd
 (m) Minz
 (b) Bunco
> b
Let's play the game of Bunco!
How many players (2-4)?
> 2
What is the name of player #1?
> Bill
There is no player named Bill.
What is the name of player #1?
> Alan
How many coins would you bid Alan (1-70)?
> 100
Invalid number of coins.
How many coins would you bid Alan (1-70)?
> 20
What is the name of player #2?
> Alan
Alan is already in the game.
```

```
What is the name of player #2?
> Steve
How many coins would you bid Steve (1-100)?
> 25
```

After the players and their bids are all set, the game will start. Games that involve multiple players are played in a round-robin manner where each player takes a turn. When needing to throw dice, the player is asked for how strong s/he would throw, choosing a number between 0 to 5 (inclusive), which will affect the outcome by shifting the face up value. For example, if the random number generated by the system was 5, and the player's input was 2, the face up value of the die will be 1. The outcome of throwing dice should be printed using the following symbols:

⚀ ⚁ ⚂ ⚃ ⚄ ⚅

The game of Even-or-Odd has only one player playing with a die. To win, the player has to guess if the die will turn up with an even or odd number. Below are some examples of how it will be played:

```
Hey Steve, Even (e) or Odd (o)?
> a
Invalid choice.
Even (e) or Odd (o)?
> o
How strong will you throw (0-5)?
> 7
Invalid choice.
How strong will you throw (0-5)?
> 3
⚄
Congratulations, Steve! You win!
```

```
Hey Alan, Even (e) or Odd (o)?
> e
How strong will you throw (0-5)?
> 1
⚀
Sorry, Alan! You lose!
```

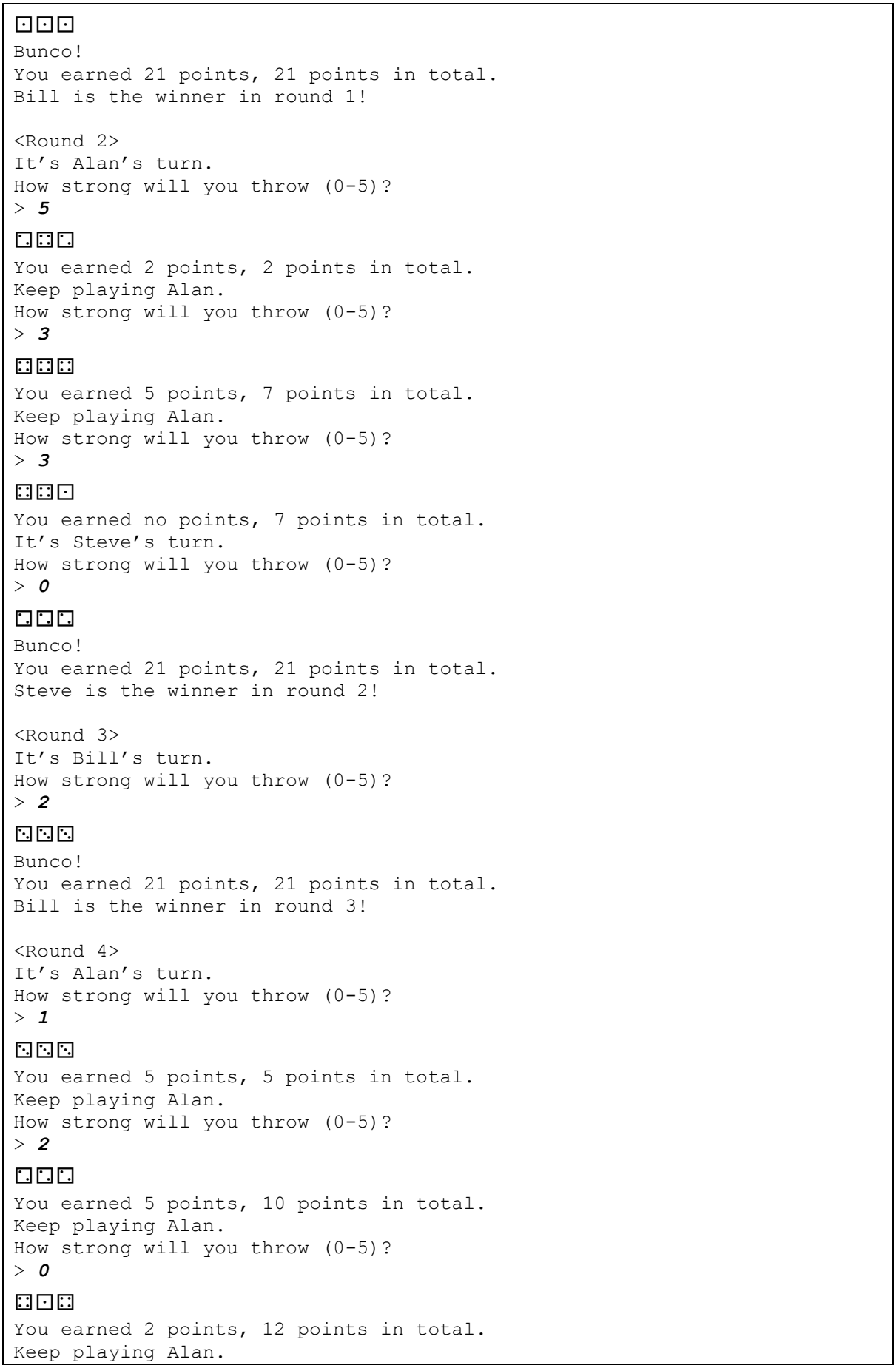
The game of Minz is played with a pair of dice. Each player in the game takes turn to throw a pair of dice and the player who gets the lowest sum of the face up values. If more than one player throws the same lowest sum, then those players keep playing with others being left out. For example, below is an example of three players playing:

```
Let the game begin!
It's Steve's turn.
How strong will you throw (0-5)?
> 3
⚀⚁
It's Alan's turn.
How strong will you throw (0-5)?
> 0
⚄⚄
It's Bill's turn.
How strong will you throw (0-5)?
```

```
> 5
⚀⚁
Players remaining: Steve, Bill
It's Steve's turn.
How strong will you throw (0-5)?
> 1
⚄⚅
It's Bill's turn.
How strong will you throw (0-5)?
> 5
⚂⚅
Congratulations, Steve! You win!
```

The game of Bunco is played with a set of three dice. There are six rounds in each game, and in each round the players take turns to roll dice in round robin (in the order of the players added to the game). The first round starts from the first player added to the game, then the rest of the rounds starts from the next player after the last player played in the previous round. Once a player in turn rolls dice, one point is awarded for each die that matches the current round number. For example, if two dice has a face up value of 3 after rolled in round 3 that player is awarded 2 points. If all three dice match the current round number (named a "Bunco"), 21 points are awarded. If all three dice match each other but do not match the current round number, 5 points are awarded. If any points are scored in his/her turn, the player gets to roll again, continuing to add to their score in round. If no points are awarded after rolling dice, that player's turn ends and the next player plays. Each round ends when a player has scored 21 points, which makes rolling a bunco an instant win. At the end of each round, the winner of that round is announced. The game ends when all six rounds are complete. At the end of the game a summary of the scores of each player in each round and the number of buncos played is reported, along with the winner of the game. The player with the most rounds won is the overall game winner, with ties broken by comparing total points scored. If the total points scored are the same, the player who had more Buncos win. Below is an example of three players playing the game of Bunco.

```
<Round 1>
It's Steve's turn.
How strong will you throw (0-5)?
> 5
⚄⚀⚂
You earned 1 point, 1 point in total.
Keep playing Steve.
How strong will you throw (0-5)?
> 3
⚀⚀⚃
You earned 2 points, 3 points in total.
Keep playing Steve.
How strong will you throw (0-5)?
> 1
⚂⚄⚁
You earned no points, 3 points in total.
It's Bill's turn.
How strong will you throw (0-5)?
> 2
```



```
⚀⚀⚀
Bunco!
You earned 21 points, 21 points in total.
Bill is the winner in round 1!

<Round 2>
It's Alan's turn.
How strong will you throw (0-5)?
> 5
⚁⚁⚁
You earned 2 points, 2 points in total.
Keep playing Alan.
How strong will you throw (0-5)?
> 3
⚂⚂⚂
You earned 5 points, 7 points in total.
Keep playing Alan.
How strong will you throw (0-5)?
> 3
⚂⚂⚀
You earned no points, 7 points in total.
It's Steve's turn.
How strong will you throw (0-5)?
> 0
⚁⚁⚁
Bunco!
You earned 21 points, 21 points in total.
Steve is the winner in round 2!

<Round 3>
It's Bill's turn.
How strong will you throw (0-5)?
> 2
⚃⚃⚃
Bunco!
You earned 21 points, 21 points in total.
Bill is the winner in round 3!

<Round 4>
It's Alan's turn.
How strong will you throw (0-5)?
> 1
⚃⚃⚃
You earned 5 points, 5 points in total.
Keep playing Alan.
How strong will you throw (0-5)?
> 2
⚁⚁⚁
You earned 5 points, 10 points in total.
Keep playing Alan.
How strong will you throw (0-5)?
> 0
⚂⚀⚂
You earned 2 points, 12 points in total.
Keep playing Alan.
```

```
How strong will you throw (0-5)?
> 3
```

```
You earned 5 points, 17 points in total.
Keep playing Alan.
How strong will you throw (0-5)?
> 2
```

```
You earned 5 points, 22 points in total.
Alan is the winner in round 4!

<Round 5>
It's Steve's turn.
How strong will you throw (0-5)?
> 4
```

```
You earned 0 points, 0 points in total.
It's Bill's turn.
How strong will you throw (0-5)?
> 2
```

```
You earned 2 points, 2 points in total.
Keep playing Bill.
How strong will you throw (0-5)?
> 5
```

```
You earned 0 points, 2 points in total.
It's Alan's turn.
How strong will you throw (0-5)?
> 5
```

```
Bunco!
You earned 21 points, 21 points in total.
Alan is the winner in round 5!

<Round 6>
It's Steve's turn.
How strong will you throw (0-5)?
> 4
```

```
You earned 0 points, 0 points in total.
It's Bill's turn.
How strong will you throw (0-5)?
> 3
```

```
You earned 21 points, 21 points in total.
Bill is the winner in round 6!

=====================================
Round      Steve       Bill       Alan
=====================================
    1          3         21          0
    2         21          0          7
    3          0         21          0
    4          0          0         22
    5          0          2         21
```

```
    6          0         21          0
=====================================
Total         24         65         50
=====================================
Bunco          1          3          1
=====================================
Bill won 3 rounds, scoring 65 points, with 3 Buncos.
Congratulations, Bill! You win!
```

Users can check how many coins they have by entering ‘c’ as shown in the example below. Additionally, users can also view the leader board to keep track of their performance. The leader board will show the list of all registered players with their details, including their name, the number of games played, the number of games won, and the number of coins remaining. The list must be sorted in the order of the number of coins, and if having the same number of coins, then in the order of winning rate (i.e., the number of games won divided by the number of games played).

```
What would you like to do?
 (n) register a new player
 (c) show your coins
 (s) show the leader board
 (p) play a game
 (q) quit
> c

You have 170 coins!

What would you like to do?
 (n) register a new player
 (c) show your coins
 (s) show the leader board
 (p) play a game
 (q) quit
> s
============================
Name       Played  Won  Coins
============================
Alan           11    8    325
Steve           5    4    170
Bill            7    3    170
Elon           15    8     95
============================

What would you like to do?
 (n) register a new user
 (s) show the score board
 (p) play a game
 (q) quit
> q

Thank you for playing Tons-o-Dice!
```

## Additional Requirements

The aim of the assignment is to allow you to practise creating an object-oriented design and implementing it in Python using object-oriented programming concepts such as *abstraction, encapsulation, class inheritance*, *polymorphism*.

In this first part of the assignment, you will need to ***develop a class design using UML class diagrams*** in UMLet based on the requirements above and in the remainder of this specification. The focus of the assignment is on identifying classes and applying appropriate relationships between the classes. **The UML class diagram MUST include all necessary classes, attributes and methods, and all relationships between classes. Also, visibility, data types, and cardinalities must be specified in detail.**

When designing the UML class diagram, you must keep in mind that the design is going to be implemented in the second part of the assignment. The UML design should be designed in such a way that code is reused and can be extended easily. Your design must presume the following Python code will run the application software you developed:

```
tod = TonsoDice()

tod.run()
```

Think of how this `run()` method will be the starting point and call methods of other objects in the system to identify classes, their methods and attributes, and their relationships based on reading the assignment task specification.

While more detailed instructions on implementation will be provided with the specification for Part 2 of this Assignment, the description in this document should provide enough details for designing the UML class diagram. However, if you have any doubt or question while developing the UML class diagram, please contact the Course Facilitator for clarification through either online forum on the course website or through email.

## Work Plan

To complete the first part of this assignment, you will need to perform the following steps:

1. Read the assignment specification carefully.
2. Review the relationships between classes in a UML class diagram: inheritance, association, aggregation, and composition.
3. Identify which classes are necessary to capture the requirements.
4. Identify attributes and methods of those classes. Specify visibility for them as well as data types for attributes, parameters and return value of each method.
5. Identify relationships between classes and add cardinalities and labels to the relationship where applicable.
6. Revisit all classes and see if you can use inheritance to minimize duplication of code.
7. If you have an abstract class, then make sure you set its name in italic.
8. You may add annotations to classes if it helps you later with the implementation (e.g., responsibilities of classes).

Identifying all methods and attributes does not have to be complete at this point in time working on the Part 1 of the assignmen. However, enough details should be captured in the UML class diagram as it should give you the opportunity to think about the structure and flow of the application before starting the implementation. You can revise it later in Part 2 of the assignment as you progress through the implementation, for example, you may realise you are missing some arguments to a member function or missing an entire function. The marking of this aspect will focus on the appropriate classes and relationships, separation of concerns, and good naming that captures the requirements and concepts described in this assignment specification document.

After completing the design, make sure to include a note with your student details (Full name and Student ID) in the diagram, then save it into a **uxf** file and submit it to the submission link on the course website.

## Submission Details

You MUST submit your UML Class Diagram in a **uxf** file to the submission link on the course website. The UML class diagram MUST be generated from the UMLet or UMLetino modelling tool: a photo of a hand-drawn diagram will NOT be accepted. The submission must include a note with your student details (Full name and Student ID).

The submission is **due by the date and time specified on the submission link on the course website**.

This assignment is an **individual task** that will require an **individual submission.**

There will be **no extensions or late submissions for this course without one of the following exceptions:**

1. A medical certificate is provided that has the timing and duration of the illness and an opinion on how much the student's ability to perform has been compromised by the illness. Please note if this information is not provided the medical certificate WILL NOT BE ACCEPTED. Late assessment items will not be accepted unless a medical certificate is presented to the Course Facilitator. The certificate must be produced as soon as possible and must cover the dates during which the assessment was to be attempted. In the case where you have a valid medical certificate, the due date will be extended by the number of days stated on the certificate up to five working days.
2. A Learning and Teaching Unit councillor contacts the Course Facilitator on your behalf requesting an extension. Normally you would use this if you have events outside your control adversely affecting your course work.
3. Unexpected work commitments. In this case, you will need to attach a letter from your work supervisor with your application stating the impact on your ability to complete your assessment.
4. Military obligations with proof.

Applications for **extensions must be lodged with the Course Facilitator before the due date** of the assignment, or as soon as possible after the incident took place.

Note: Equipment failure, loss of data, 'Heavy work commitments' or late starting of the course are not sufficient grounds for an extension. Make sure you make a back-up of your **.uxf** file frequently, especially when working on a web-based UMLetino.

## Marking Criteria

Your UML Class Diagram will be marked against an expected design. The focus is on the correct identification of classes and their relationships, and capturing enough details from this assignment specification. Penalties will be applied if the design is not submitted in the correct format.

| Criteria | Mark |
|---|---|
| Correct identification of classes | 20 |
| Attributes with datatypes and visibility | 10 |
| Methods with parameters, return data types, and visibility. | 10 |
| Correct use of inheritance. | 20 |
| Correct use of aggregation and/or composition with cardinalities. | 20 |
| Correct use of association with labels and reading directions. | 20 |
| Submission not in appropriate format or missing student details.<br>(Only UMLet/UMLettino drawn diagram **including student details** in a **uxf** file is acceptable.) | Up to -15 |
| **Total Possible Marks** | **100** |

## Academic Misconduct

**This assignment is an individual task that will require an individual submission.**

Students are reminded that they should be aware of the academic misconduct guidelines available from the University of South Australia website.

Deliberate academic misconduct, such as plagiarism, is subject to penalties. Information about Academic integrity can be found in Section 9 of the Assessment Policies and Procedures Manual at:

https://i.unisa.edu.au/policies-and-procedures/codes/assessment-policies/