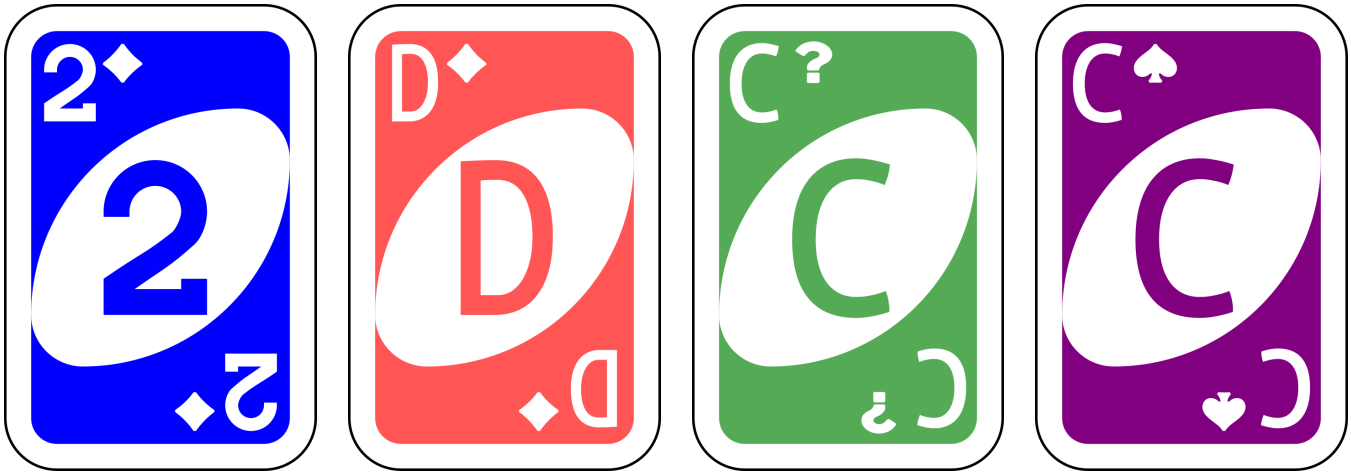
Assignment 2: Final Card-Down

This is a team assignment, to be done with your assignment 2 group.

This assignment is due on **Sunday 29th October**, at **23:59:59** (Sydney time).

The game *Final Card-Down* is a card game played with a special printed deck.

Each card has a *value* between 0x0 and 0xF, a *suit* of either hearts, diamonds, clubs, spades, or questions, and a *color* of either red, green, blue, yellow, or purple.

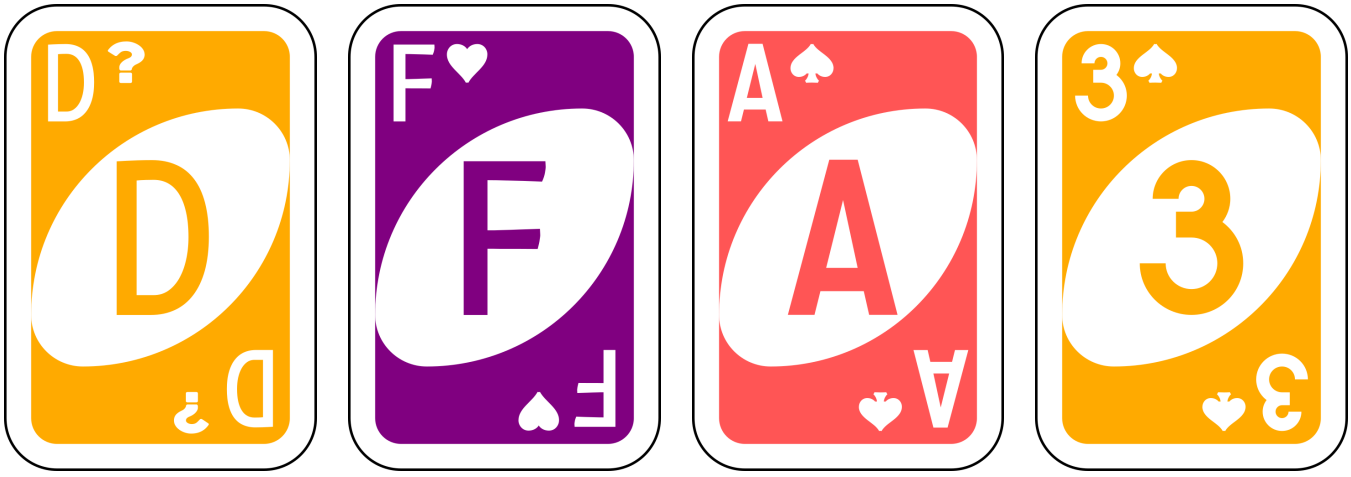


Each *player* is dealt a *hand* of 7 cards. The first *player* to place their *final card down* in the discard pile wins.

For more details about the *game*, see the **Final Card-Down Rules**

Objectives

For this assignment, not only will you be implementing a card game, but you'll also be writing an *artificial intelligence* to play the game.



1. Implement a set of *tests* for an *Abstract Data Type* (ADT) that represents the rules of a card game. This is to be done in your **assignment group** of 4 students.
2. Implement the ADT and the rules of the card game, making sure that is passes all of the **tests**. This is to be done in your **assignment group** of 4 students.
3. Write a AI to play the game for you. This is to be done in your **lab pair** of 2 students.

Before you begin...

Working with your group

In this assignment you will be working with 3 other people. You must make sure that you can keep in contact with them. Arrange a time to meet regularly each week and work on the assignment, outside of tutorials.

A good way to keep up to date with your group is to meet up for lunch at least once a week and discuss the assignment. Talk about what you're working on, what your having trouble understanding, what you need help from your team with, and how your solving the problems you find in the assignment.

Every member of the group is expected to make a contribution to every part of the solution. Every member of the group must work on **both** parts; writing the tests and implementing the ADT.

With your lab partner, you must **both** work in implementing your AI.

> You must write regular, individual **progress blogs** as you work on the project.

Making a plan

Make a plan about how to approach the assignment. Don't assume that you can do it all in one go. Break it up into parts and make sure you write tests for anything before you implement it.

Set yourself goals, like having a certain set of functions implemented and tested by a particular date.

Make a plan about how you will implement the ADT, and break up the code between your group members.

Make sure to review the code of your team members regularly and give feedback on how it could be improved to have better style and to fix any bugs you might find.

> You must individually write a **blog** with your plan before you start writing any code. You will be provided with a blog template, as with the Mandelbrot assignment.

Understanding the problem

Read through **the Game Rules** to get a better understanding of how the game works and how you might represent it with code.

Then, read carefully through the **.h** files and try to figure out what information you will need to store and how you might want to store it in your ADT.

Get the files

There are three header files you'll need for this assignment:

- **Card.h**
- **Game.h**
- **player.h**

You shouldn't modify these header files! If you do, your family will be haunted by semicolons for generations!. Instead, you should declare any additional constants, structures, and prototypes in the .c file that implements them. Make sure you mark your own helper functions `static`.

The files that you need to create are:

- **testGame.c**, which tests an implementation of the functions in **Game.h**,
- **Game.c**, which implements the functions described in **Game.h**, and
- **player.c**, which implements an AI using the function described in **player.h**.

You will also need to modify your **Card.c** from the lab to implement the interface in **Card.h**, but you will not need to submit a **Card.c**. In the **Card.c** from the lab you implemented a card where the value was represented as a `char`, but the version of `Card` in the game represents the value of the card using an new `enum` type.

The Task

Testing the ADT

This section will be released first, before week 10.

You will start by writing a simple set of tests first. The tests should be implemented in **testGame.c** and test the implementation of the functions in **Game.h**. This will help you get an understanding of how the game works. You should try and submit a basic set of tests as soon as possible and work on improving the tests you write as you get closer to the due date. You will need to write tests continually throughout the assignment period. Don't simply stop writing tests as soon as you start on the implementation.

> It is incorrect to write tests using inputs that are invalid for a function.

To run the tests yourself, use the command:

```
$ dcc -o testGame testGame.c Game.c Card.c
$ ./testGame
All tests passed. You are awesome!
```

Writing the Game ADT

This section will be released during week 10.

Once you have written some tests and submitted them, you can start working on your Game ADT. It is important that you continue to write and improve your tests as you work on your Game implementation. This will help you ensure that when you find bugs in your code they get removed for good and as your code gets more complex you can still be sure that it is correct.

Your game implementation should be implemented in **Game.c** and implement the functions from **Game.h**. Don't try and implement the ADT in one go, but try and break it up into parts, making sure you have tests for each part before trying to implement it.

Try and submit a basic implementation of the ADT as soon as possible.

The AI

This section will be released during week 11.

With your **lab partner**, you will also need to implement an AI to play the game for you. You must implement the function found in **player.h** and your AI can use any of the functions from **Game.h**, except those which specify that they should not be used by the AI, to determine what move it should make.

Your player should not take more than a few seconds to decide its move. If your AI takes too long to decide on its action, the game will end and your AI will be *at fault*.

Testing

Three times a week, until the assignment due date, every submitted **Game.c** and **testGame.c** from every group will be collected and used to test each other. This means that not only must your implementation pass your own tests, but it must pass the tests of every other group in the course. The tests you write must also be correct, and must pass ADTs that are implemented correctly and fail ADTs that are implemented incorrectly.

> You **must not** share your code with any other group in the course.

Every few days, we will run SPOTS to test all of the submissions. You will then be able to log onto a website and see how many tests you passed and how many tests you failed. You will also be able to see why your ADT failed tests. This will help you as you implement your ADT to ensure it is implemented correctly.

You may find that you failed a test, but the test was incorrect, not your implementation. If you think the test is incorrect, you can *dispute it* and tell the group that wrote the test why it is incorrect. The group that wrote the test may then disagree with you and suggest why their test is correct. If both groups think that the other group is at fault, a tutor will determine who is correct.

This means that every few days you can log into SPOTS and see how you need to improve your implementation or how to improve your tests.

Competition

Every few days, when tests are run on all of the submissions, we will also hold a competition. Each competition will have your AI play in games against 3 other AIs. At the end of each competition the aim is to score the fewest total points over all the games your AI played in.

A game can end in one of the following ways:

1. A player plays their final card, winning the game,
2. A player is unable to make a move and there is no way to draw a card (such as both the draw and discard piles being empty),
3. A player takes too long to make a move,
4. A player tries to make an illegal action, or
5. A player tries to use a function that an AI should not access.

If a player causes to end the game in any way other than by winning the game or a player being unable to draw a card, they will be considered *at fault*.

Competition results will be available from SPOTS.

Submitting

Your assignment group should make a joint submission for your *testGame.c* and your *Game.c*. These should be submitted to the *testGame* and *Game* activities.

Your AI pair should make a joint submission for your *player.c*. This should be submitted to the *player* activity.

As long as your assignment group and AI pair are both set up correctly, only one of you needs to submit in order for all members of that group to see your submission.

Submit as you usually would from a CSE system —

```
$ give cs1511 testGame
$ give cs1511 Game
$ give cs1511 player
```

— or to the relevant activity on Give Online.

You should submit often, for your Game ADT and your AI to play in the nightly competition rounds!

Assessment

This assignment will contribute 15% to your final mark.

Mark breakdown

> Your combined mark for correctness + performance is capped at **60%**.

25%	**testGame.c** correctness
25%	**Game.c** correctness
15%	**player.c** performance
20%	Code style
20%	Planning, process, and reflection blogs

You may have noticed that the percentages above add to more than 100%. This is because your combined mark for correctness+performance (testGame.c, Game.c, player.c) is capped at 60%, which means that you can still get full marks for correctness without needing your AI to be at the top of the leader-board.

Plagiarism

Note: although this is a group assessment, plagiarism rules still apply.

Penalty	Description
-70%	Knowingly providing your work to anyone (outside of your group) and it is subsequently submitted (by anyone).
-70%	Submitting any other person's work (outside of your group). This includes joint work (across several groups).
0 FL for COMP1511	Paying another person to complete work. Submitting another person's work without their consent.

Keep in mind that if you "cheat" on this assignment, the only person you're cheating is yourself. We give you these assignments so that you can gain experience and develop new skills. If you plagiarise, you're throwing away a great opportunity, and letting down your team-mates.

Late Penalty

The assignment is due at **23:59:59** on **Sunday 29th October**, Sydney time.

You will not be able to submit late, as a part of the assessment is of your time and project management.

It is essential that you consistently work on the assignment, and regularly submit the progress you have thus far.

This is not an assignment that you can just rush through immediately before the due date: you will need to demonstrate consistent, incremental improvement throughout the period of the assignment.

(This is also a University policy, as this is the latest possible due date for an assessment task in the teaching period.)

Resources

ChangeLog

- **v1.2.2** (2017-10-24):
  - Made due dates consistently Sunday Week 13.
- **v1.2.1** (2017-10-09):
  - Clarified behaviour and use of `playerPoints` function.
  - Clarified when the game ends.
  - Players may only call out a previous player once for each of `SAY_UNO`, `SAY_DUO`, `SAY_TRIO`.
- **v1.2.0** (2017-10-09):

    - Added `topDiscard` to get the card on top of the discard pile.
- **v1.1.3** (2017-10-03):
    - Added submission directions.
- **v1.1.2** (2017-10-02):
    - Renamed `pastTurns` to `numTurns`, and clarified what it should do.
- **v1.1.1** (2017-09-30):
    - Added `gameWinner` to find if the game has ended
    - Clarify game ending under odd circumstances.
    - Clarify end of turn.
- **v1.1.0** (2017-09-28):
    - Changed `getPlayDirection` to `playDirection`.
    - Added `NUM_PLAYERS` constant.
- **v1.0.0** (2017-09-27): Released to unsuspecting students.