

Assignment 2: Testing the Game ADT

An Introduction to Testing the Game ADT

sampleTest.c

Do not copy this verbatim into your ADT tests.

Do not copy the code from this into your ADT tests.

Use it as a reference to understand how to go about creating a game, setting up the game state, and using it to write tests.

Do not copy this code and use it in your assignment.

```

// A sample program to demonstrate testing the Final Card-Down Game ADT
// Andrew Bennett 2017-10-02

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

// You should make various functions to put each of your tests in.
// Don't just put the tests directly in the main function :(
void basicCurrentPlayerAndTurn (void);

int main (int argc, char *argv[]) {

    // You might want to print something out at the start of your tests:
    printf ("Hello and welcome to GROUP_NAME's Game ADT tests!\n");

    // Call your various tests here.

    // So far, we just have the one test:
    basicCurrentPlayerAndTurn ();

    // And, of course....
    printf ("All tests passed, you are Awesome!\n");

    return EXIT_SUCCESS;
}

// (It's important to have a comment at the start of each function,
// giving a brief overview of what it does... like this one:)
//
// Tests currentPlayer and currentTurn, by constructing a very simple
// game, making moves as each player, and checking that the
// current player and current turn are correctly updated.
void basicCurrentPlayerAndTurn (void) {

    // The way that we test an ADT as complicated as the Game ADT is by
    // constructing a game, and effectively playing out a game -- by
    // making moves, and checking at each point that the ADT functions
    // give us back the values that we expect.
    //
    // Since we have full control over the deck that's used in the game,
    // we can make whatever decks we like in order to test whatever
    // attributes we want.
    //
    // You might want to make a very simple and contrived deck to test
    // functions initially, e.g. a deck that consists of 50 copies of
    // the same card (even though it would make a very boring game to
    // actually play as players).
    //

```

```

// Later on, you should make more interesting and more complicated
// decks, to test the ADT more thoroughly -- it's very possible that
// there are bugs in the implementation that you wouldn't pick up
// just using a deck with 50 of the same card.
//
// Once you've written fairly thorough tests for each ADT function,
// you might want to make some overall tests from "real games" --
// actually play out a game, with an interesting / "real" deck, and
// record the moves that you made when actually playing the game as
// a human, and code that up as another test in your testGame.c
// file.
//
// For now, though, let's just stick with a very simple test, to
// illustrate how to get started with testing the Game ADT.

// You might want to print something out at the start of each of
// your test functions, to help see what's going on when you look at
// the output in the terminal later on.
printf ("A very basic test for currentPlayer and currentTurn:\n");
printf ("This test has a deck of 50 cards, all are RED 1 HEARTS\n");

// The newGame prototype looks like this:
//
//      Game newGame (int deckSize,
//                    value values[], color colors[], suit suits[]);
//
// In order to test it, we need to specify the cards that will be
// used in the game -- i.e. the cards that will make up the game's
// deck.
//
// We do this by passing in arrays that specify the value, color,
// and suit of each card. The value/color/suit in array index 0 will
// be the first card on top of the deck, the value/color/suit in
// array index 1 will be the second card of the deck, below the
// first, etc etc.
//
// We need to make arrays of values, suits, and colors. To keep
// things simple, we'll make a deck that only has the same type of
// card: a RED 1 of HEARTS.
//
// This means we'll need to make an array of values of size 50,
// wherein all of the values are 1:
// e.g. value values[50] = {1, 1, 1, 1, 1, ....}; etc for 50 1s.
// To make it nicer, we can just initialise an array that's 50 big,
// and then go through and fill it in a loop.
//
// We'll set all three arrays up at once, to save time/code:
value values[50];

```

```

suit suits[50];
color colors[50];

int i = 0;
while (i < 50) {
    values[i] = 1;
    suits[i] = HEARTS;
    colors[i] = RED;
    i++;
}

// We now have our arrays prepared, time to make a new game!
Game game = newGame (50, values, colors, suits);

// Now we have a game object called `game`, which we can use to play
// the game.
//
// At this point, we're at the very start of a new game,
// so the current player is 0, and the current turn is 0.
// These are things we can test!
//
// It's nice to print out what you're testing; it's up to you
// whether you number the tests, or whether you just describe what
// you're about to test. Be consistent!
printf ("Test 1: Checking that the current player is 0\n");
assert (currentPlayer (game) == 0);

printf ("Test 2: Checking that the current turn is player 0\n");
assert (currentTurn (game) == 0);

// Since we've constructed the deck, we know exactly which cards
// will be where (they'll all be the same card, so we don't even
// have to think about it).
//
// If our deck were made of different types of cards, so it was a
// bit more interesting, we'd still be able to tell which cards
// should be where: the top card on the deck (from index 0 of the
// three arrays we passed in) will go to Player 0, the second card
// on the deck (index 1) will go to Player 1, the third to Player 2,
// the fourth to Player 3, the fifth to Player 0, etc etc.
//
// Once we've dealt the cards to all of the players (7 cards per
// player * 4 players = 28 cards), we take the next card on the
// top of the deck, and that becomes the start of our discard pile.
//
// This is the card that players have to match as they go around the
// table and play cards: player 0's card has to match the card we've

```

```

// just made into the discard pile in either suit, color, or value.
//
// Player 1's card has to match the card that player 0 played, in
// either suit, color, or value.
//
// Since all of the cards in the deck are the same, we know that
// every player can just play the first card in their hand, and it
// will always be a valid move.
//
// Using a simplified deck like this makes it very easy to test
// that things like the turn number and current player are being set
// correctly in the ADT.
//
// So, let's start out by making a move as the first player
// (Player 0)

// As mentioned earlier -- every card in each player's hand is valid
// to play at any point during this game.
//
// This means that we can just get the first card from the current
// player's hand, and play that card, then it becomes the next
// player's turn, and we can play the first card in their hand,
// etc., until the game ends.
//
// So, in order to make a move as a player, first we need to
// actually get a card from their hand. We can use the `handCard`
// function for this: it will give us the card from the current
// player's hand at the specified position.
//
// In this case, we'll just take the first card in their hand (which
// will be the card at position 0, since we count from 0) -- we know
// that this will be a RED 1 of HEARTS.
Card toPlay = handCard (game, 0);

// We then want to make a move wherein we play that card.
// In order to do that, first we need to create a move struct, that
// describes the move that we want to make.
//
// The move struct is defined in Game.h (the actual struct fields
// etc are defined in Game.h, not just a typedef to a pointer to it
// like we have with the _game struct). This means that the move
// struct is a concrete type, not an ADT, and we can just fiddle
// around directly with the fields inside it.
//
// First, we need to make a move struct:
playerMove move;

```

```

// Now we have a struct called `move`, which is in the memory for
// this function (on the "stack"). We haven't dynamically allocated
// it, so it's not on the heap, we literally have the struct itself
// sitting in the stack frame for this function. (This means we
// don't have to free anything later)
//
// The move struct has three fields:
// - `action` -- this is the action to play;
// - `color` -- only valid when playing a DECLARE; and
// - `Card` -- which card to play; this is only valid when
//   playing a card.
//
// So, since we want to play a card, we need to set the action type
// to be PLAY_CARD:
move.action = PLAY_CARD;

// And we need to set the card to be the card that we want to play.
// We got this card from the player's hand before
// (or rather, we got a Card -- a pointer to a card struct)
// and this is what we want to pass in to the Game ADT's playMove
// function when we make our move.
move.card = toPlay;

// In order to actually play the move, we call the playMove function
playMove (game, move);

// Woohoo, we've made our first move!
//
// Within the Game ADT, the state being tracked -- the cards, the
// players, the turns -- should all change to reflect the move we
// made. You should use other ADT getters to check that the move
// happened, and the changes you expect have been made.
//
// In this case, because we played a card, the card should now no
// longer be in the player's hand; instead it should be on the top
// of the discard pile.

// Finally, we want to end player 0's turn; we do this by sending
// the END_TURN move to playMove:
playerMove finalMove;
finalMove.action = END_TURN;
playMove (game, finalMove);

// Now that Player 0 has played the END_TURN move, the game moves
// on, and it becomes Player 1's turn.
//
// At this point, it should now be turn 1 of the game, and the

```

```
// current player should now be player 1:
printf ("Testing that currentPlayer is Player 1\n");
assert (currentPlayer (game) == 1);
printf ("Testing that currentTurn is 1\n");
assert (currentTurn (game) == 1);

// Now that we've moved on to Player 1's turn, we should play a move
// as player 1 (playing a card, again, then ending the turn),
// followed by playing a move as player 2, then player 3, then
// player 0 again, etc etc until the end of the game.

// Once you've hit the end of your function, don't forget....
printf ("All tests for currentPlayer / currentTurn passed!\n");
}
```

Do not copy this code and use it in your assignment.

But do use it as a guide to help you understand how to go about testing the Game ADT.

Good luck with your tests!