Τεχνολογία Λογισμικού

4ο ΠΑΡΑΔΟΤΕΟ

Test-cases-v1.0

# "Χτίσε την Αποικία Σου"


| Ονοματεπώνυμο | Αριθμός μητρώου | Έτος φοίτησης |
|---|---|---|
| Παναγόπουλος Ευστάθιος | 1072571 | 6ο |
| Δέμης Μιχαήλ | 1080958 | 6ο |
| Μυλωνάς Ανδρέας | 1084619 | 5ο |
| Κλαυδιανος Ευθύμιος | 1072592 | 6ο |
| Γιαλένιος Αριστείδης | 1072556 | 6ο |


**Test-cases1: Τοποθέτηση σπιτιού**

Ελέγχει  όταν καλούμε τη place_settlement(), τότε:

- Ένα σπίτι τοποθετείται σωστά πάνω σε ένα εξάγωνο του χάρτη.
- Το νέο σπίτι ανήκει στον παίκτη.
- Το σπίτι  τοποθετείται στη σωστή θέση

**Code:**

```
import pytest
from main import Game, Player

def test_place_settlement_adds_to_player():
    game = Game()
    player = Player("Tester", (255, 0, 0))
    game.players = [player]
    game.current_player_index = 0

    tile = game.tiles[0]
    pos = tile.vertices[0]

    assert len(player.settlements) == 0
```

```
    game.place_settlement(pos)


    assert len(player.settlements) == 1
    assert any(game.is_close(s.location, pos) for s in player.settlements)
```

**Test-cases2: έλεγχος ζαριού**

- Δημιουργεί νέο παιχνίδι και έναν παίκτη.
- Εκτελεί roll_dice() για τον παίκτη.
- Ελέγχει ότι το άθροισμα των δύο ζαριών (game.last_roll) είναι μέσα στο σωστό εύρος [2–12]**.**
- Αν όχι, εμφανίζει μήνυμα λάθους με την τιμή που βρέθηκε.

**Code:**

```
import pytest
from main import Game, Player


def test_roll_dice_range():
    game = Game()
    player = Player("Tester", (255, 0, 0))
    game.players = [player]
    game.current_player_index = 0


    game.roll_dice()


    assert 2 <= game.last_roll <= 12, f"Invalid dice roll: {game.last_roll}"
```

**Test-cases3: έλεγχος παράδοσης αγαθών**

- Φτιάχνει ένα πλασματικό tile με number=8 και resource='wood'
- Τοποθετεί ένα οικισμό πάνω σε κορυφή του tile
- Καλεί χειροκίνητα distribute_resources() αφού θέσει game.last_roll = 8
- Ελέγχει αν ο παίκτης έλαβε 1 μονάδα από τον αντίστοιχο πόρο

**Code:**

```
import pytest
from main import Game, Player, Settlement
```

```
def test_roll_dice_gives_resources():
    game = Game()
    player = Player("Tester", (255, 0, 0))
    game.players = [player]
    game.current_player_index = 0

    # Χειροκίνητα ορίζουμε ένα tile με known number (π.χ. 8)
    test_tile = game.tiles[0]
    test_tile.number = 8
    test_tile.resource_type = 'wood'

    # Βάζουμε settlement δίπλα στο tile
    pos = test_tile.vertices[0]
    settlement = Settlement(player, pos)
    test_tile.settlements.append(settlement)
    player.settlements.append(settlement)

    # Δίνουμε roll που ταιριάζει με το tile number
    game.last_roll = 8
    game.distribute_resources()

    # Ελέγχουμε ότι ο παίκτης πήρε 1 wood
    assert player.resources['wood'] >= 1, "Ο παίκτης δεν πήρε τον πόρο!"
```

**Test-cases4: έλεγχος κατασκευής κάρτας μαγείας**

- Ότι καταναλώνονται οι 3 απαιτούμενοι πόροι (sheep, wheat, ore)
- Ότι ο παίκτης παίρνει είτε πόντο νίκης είτε στρατιώτη

**Code:**

```
import pytest
from main import Game, Player

def test_buy_card_consumes_resources_and_awards_effect():
    game = Game()
    player = Player("Tester", (255, 0, 0))
    game.players = [player]
    game.current_player_index = 0

    # Δίνουμε ακριβώς τους πόρους για αγορά κάρτας
    player.resources = {
        'wood': 0, 'brick': 0,
        'sheep': 1, 'wheat': 1, 'ore': 1
    }

    points_before = player.card_points
    knights_before = player.knights

    game.buy_card()

    # Βεβαιωνόμαστε ότι καταναλώθηκαν οι πόροι
    assert player.resources['sheep'] == 0
    assert player.resources['wheat'] == 0
    assert player.resources['ore'] == 0

    # Ελέγχουμε ότι πήρε είτε πόντο είτε στρατιώτη
    assert player.card_points > points_before or player.knights > knights_before
```

**Test-cases5: έλεγχος τοποθέτησης δρόμου**

- Ο δρόμος προστίθεται στη λίστα του παίκτη.
- Ο δρόμος αποθηκεύεται και στον συνολικό πίνακα game.roads.
- Συνδέεται σωστά με κορυφές από τα εξάγωνα του χάρτη.

**Code:**

```
import pytest
from main import Game, Player, Settlement


def test_place_road_adds_road_to_player():
    game = Game()
    player = Player("Tester", (255, 0, 0))
    game.players = [player]
    game.current_player_index = 0

    # Προσθέτουμε settlement χειροκίνητα για να μπορεί να ξεκινήσει ο δρόμος από εκεί
    tile = game.tiles[0]
    start = tile.vertices[0]
    end = tile.vertices[1]
    mid = ((start[0] + end[0]) / 2, (start[1] + end[1]) / 2)

    settlement = Settlement(player, start)
    tile.settlements.append(settlement)
    player.settlements.append(settlement)

    # Ο παίκτης έχει 2 free roads στην αρχή, οπότε δεν απαιτούνται πόροι
    assert len(player.roads) == 0
    assert len(game.roads) == 0

    game.place_road(mid)

    # Έλεγχοι
```

```
    assert len(player.roads) == 1
    assert len(game.roads) == 1

    road = player.roads[0]
    assert game.is_close(road.start_pos, start) or game.is_close(road.end_pos, start)
```

**Test-cases6: έλεγχος τοποθέτησης σπιτιού πολύ κοντά**

- Τοποθετεί έναν πρώτο οικισμό.
- Προσπαθεί να τοποθετήσει δεύτερο πολύ κοντά στον πρώτο.
- Περιμένει ότι δεν θα τοποθετηθεί

**Code:**

```
import pytest
from main import Game, Player, Settlement

def test_cannot_place_settlement_too_close():
    game = Game()
    player = Player("Tester", (255, 0, 0))
    game.players = [player]
    game.current_player_index = 0

    tile = game.tiles[0]
    pos1 = tile.vertices[0]
    pos2 = tile.vertices[1]  # συνήθως κοντινή κορυφή

    # Τοποθετούμε οικισμό στη pos1
    settlement = Settlement(player, pos1)
    tile.settlements.append(settlement)
    player.settlements.append(settlement)
```

```
    # Προσπαθούμε να τοποθετήσουμε δεύτερο πολύ κοντά
    before_count = len(player.settlements)
    game.place_settlement(pos2)
    after_count = len(player.settlements)

    # Αν το παιχνίδι δουλεύει σωστά, δεν θα προστεθεί δεύτερος
    assert after_count == before_count, "Ο παίκτης δεν θα έπρεπε να μπορεί να τοποθετήσει κοντινό οικισμό"
```