# Raport de Testare Manuala - B3 Catan Testing

## Mediu de testare

-OS: Linux Mint 21.3 Virginia (Kernel 5.15.0-91-generic, x86\_64)

- Unity Editor: 6000.0.43f1 LTS

- Scenă testată: Game.unity

- IDE: Rider (Attach to Unity Editor)

## 1. Initial Placement

La pornirea jocului, fiecare jucător trebuie să plaseze câte o așezare și un drum în ordinea crescătoare a indexului (0→1→2→3), apoi în ordinea descrescătoare (3→2→1→0).

În metoda `StartMachineWithOtherPlayer()`, apelul `StartMachine()` repornea `InitialiseState`, regenerând tabla si reinițializând jucătorii. Acest comportament crea un blocaj și crash după primul EndTurn.

Rezolvare:

În `StartMachineWithOtherPlayer()

am înlocuit: StartMachine();

cu stateMachine.ChangeState<AddHouseState>();

Astfel, nu se mai repornește fluxul de inițializare, iar plasarea inițială se efectueaza corect.

**2. Ordinea Jucatorilor dupa PlayerEndState**

După faza de plasare inițială (0→1→2→3→3→2→1→0), jocul trebuie sa reia ciclul normal (0→1→2→3). Din cauza omiterii reinițializării variabilei \_isInverseOrder, din methoda **AddRoadState.OnEnter** în blocul if:

if (stateMachine.IsFirstPlayer && \_isInverseOrder)

{

stateMachine.ChangeState<PlayerDiceGameState>();

yield break;

}

ordinea jucătorilor nu era eronată. Ca rezolvare, am completat condiția pentru a reseta flag-ul `**\_isInverseOrder**` și a reveni la ordinea normală

## 3. Distributie Resurse (roll ≠ 7)

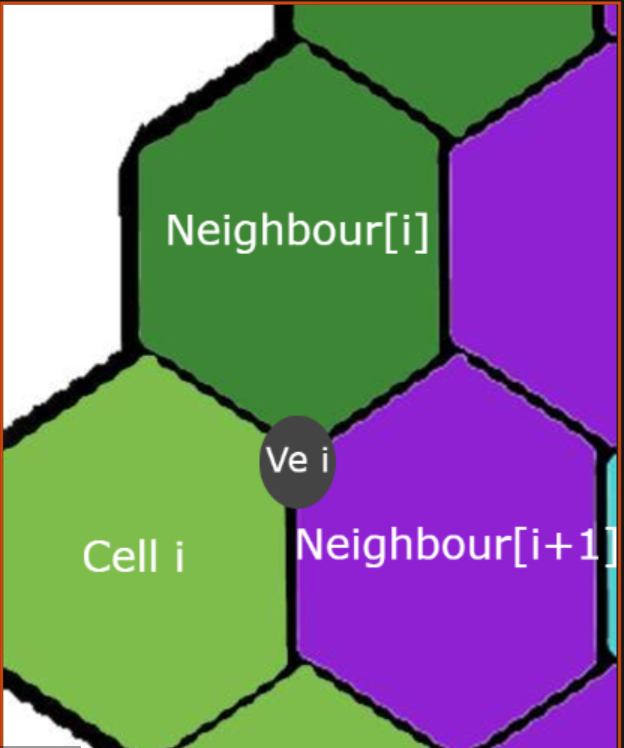
După aruncarea zarurilor a căror sumă este o valoare diferită de 7, jocul intră în **ResourceDistributionState** și distribuie resurse jucatorilor cu așezări lângă hex-urile corespunzătoare. Fluxul continuă apoi natural spre `**PlayerFreeState`.**

## 4. Construirea asezarilor si drumurilor (Building Logic)

Metodele `**BuildHouseCoroutine**` cât si **BuildRoadCoroutine** din clasa **HumanPlayer** folosesc metodele din clasa `**FullHexGrid**` pentru a obtine vertex-uri ,muchii și vecini.   
 În încercarea de a construi o casă prin metoda BuildHouseCoroutine se observă că settlement-urile sunt plasate eronat cu o probabilitate de ⅙. În **BuildHouseCoroutine** este apelată metoda RayCastCoroutine care are rolul de a trage o rază imaginarp de la cameră la punctul în care a fost apăsat și a găsi cel mai apropiat obiect de acea rază. Având obiectul găsit(va fi un hex) se caută prin metoda **getClosestCorner** cel mai apropiat colț al celulei lovite de clickul efectiv(**\_hitpoint**). Prin Loguri de debug se observă că var corners = boardGrid.GetHexVertices(hexPosition) oferă un tuplu de settlement și direcția propriu-zisă a colțului hexagonului(dreapta sus ,dreapta,dreapta jos,...) în care poziția settlementului e mereu duplicată si nu coinicde întotdeauna cu  
var cornerPosition = boardGrid.GetHexCorner(corner.Item2, hexPosition);(corner.item2 e direcția colțului hexagonului)  
 Mergând în **FullHexGrid**, în metoda public Cell this[HexPosition position] , unde sunt initialization celulele și proprietățile lor găsim for-ul acesta care caută să folosească vertexuri și edgerui ale unor vecini deja existenți   
for (int i = 0; i < 6; i++) {

edges[i] = neighbours[i]?.Edges[(i + 3) % 6];

vertices[i] = (neighbours[i]?.Vertices[(i + 2) % 6]) ?? (neighbours[(i+1) % 6]?.Vertices[(i + 3) % 6]);

}  


Logica de reutilizarea a vertex-lui vecinilui de la poziti i+1 nu era corectă `(i+3)%6`, ducând la duplicare și/sau omiterea unor colturi valide. În celula vecina i+1 pozitia nodului corespunzător de intersecție nu este i+3, ci i+4 pentru a ajunge la corespondentul corect în plan.

Rezolvare:

În **`FullHexGrid`**:

vertices[i] = neighbours[i]?.Vertices[(i + 2) % 6]

?? neighbours[(i + 1) % 6]?.Vertices[(i + 4) % 6];

## 6. Buff-uri Porturi

Există 2 tipuri de porturi

- Port universal → buff 3:1

- Port specific → buff 2:1 pentru o resursă specifică

Folosind un log de debuger am verificat adăugarea corectă a efectelor de player buff pentru comercializarea cu banca în cazul adăugărei unui settlement într-un hexagon cell care este de tipul port și omiterea oricărui efect în caz contrar

In `AddPortBuffForSettlement`:

string buffText = port.ResourceType.HasValue

? $"2:1 {port.ResourceType.Value}"

: "3:1";

Debug.Log($"[PORT DEBUG] Applying port buff {buffText} to player {player.name}");

## 7. Validare Distribuire Resurse

După adăugarea building-urilor și road-urilor, jocul face tranziția către PlayerDiceState care simulează aruncarea zarurilor de jucător cu un generator random. Bazat pe suma punctelor celor 2 zaruri, dacă este diferită de 7, jocul distribuie corect resurse jucătorilor care au construit settlement uri pe un hexagon/celulă cu numărul = cu suma respectivă.

Pe lângă debugging , logurile de verificare din `**ResourceDistributionState**` pe care le-am adăugat confirmă corectitudinea funcționalității

Debug.Log($"Resource Given to: {[owner.name](http://owner.name)}");

## 8.Construirea pe noduri port , obiecte SettlementController null, reguli construcție la 1 nod distanță

## Observație descoperită În timpul testelor de construire, am constatat că, în zona porturilor (marginea hărții), unele entry‐uri în FullHexGrid nu returnau niciun SettlementController (returnau null). Când metoda CanBuildHouse sau CanBuildRoad încerca să acceseze proprietăţi (.HasOwner, .Owner) pe acel nod null, se producea un NullReferenceException.

## În plus, am observat că, deși în Catan nu ar trebui să poți construi așezări pe nodurile aflate pe marginea cu apă, logica existentă permitea totuși plasarea unei case acolo.

## În această linie, ConnectedPortController este null pentru orice nod normal și nenul doar dacă acel SettlementController se află exact într‐o intersecție asociată unui port. Așadar se blochează în mod explicit plasarea casei pe marginea de apă.

* În jocul de Catan există o regulă esențială: clădirile (asezările) trebuie plasate la distanță mai mare de 1 nod (adică să nu existe niciun alt settlement la un nod vecin direct). În plus, orice drum (road) trebuie să fie conectat fie la un settlement al jucătorului, fie să continue un lanț de drumuri deja construite de același jucător. Implementarea curentă verifică aceste reguli în metodele CanBuildHouse (prin CheckSettlementDistanceRule) și CanBuildRoad (prin IsDirectlyConnectedToPlayerAssets), astfel încât nicio așezare ilegală nu poate fi plasată și niciun drum neconectat nu poate fi construit.

## Testare comerț jucător ↔ bancă (UnityTest / NUnit-style)

## Obiectiv: Verificarea corectă a metodei TradeResources(player, int[] resourcesGiven, int[] resourcesWanted) din TradeController, folosind scena reală („Game”) și componentele instantiate în editor.

### Setup general

## Încărcarea scenei „Game”: [UnitySetUp]

## public IEnumerator LoadGameSceneAndFindComponents()

## {

## yield return SceneManager.LoadSceneAsync("Game");

## yield return null; // un frame pentru a finaliza inițializarea scenei

## \_bankController = GameObject.FindObjectOfType<BankController>();

## \_tradeController = GameObject.FindObjectOfType<TradeController>();

## \_playerA = GameObject.FindObjectOfType<HumanPlayer>();

## // Confirm că instanțele nu sunt null

## Assert.NotNull(\_bankController);

## Assert.NotNull(\_tradeController);

## Assert.NotNull(\_playerA);

## }

### Teste specifice

#### 1. BankTrade\_Default4to1\_UpdatesPlayerAndBankCorrectly

## [UnityTest]

## public IEnumerator BankTrade\_Default4to1\_UpdatesPlayerAndBankCorrectly()

## {

## int woodIdx = (int)ResourceType.Wood;

## int brickIdx = (int)ResourceType.Brick;

## 

## // Asigurăm resurse inițiale

## int beforePlayerWood = \_playerA.Resources[woodIdx]; // ex. 99

## int beforePlayerBrick = \_playerA.Resources[brickIdx]; // ex. 99

## int beforeBankWood = \_bankController.CurrentResources[woodIdx];

## int beforeBankBrick = \_bankController.CurrentResources[brickIdx];

## // Pregătim ofertele

## int[] resourcesGiven = new int[5];

## int[] resourcesWanted = new int[5];

## resourcesGiven[woodIdx] = 4;

## resourcesWanted[brickIdx] = 1;

## // Executăm tranzacția

## \_tradeController.TradeResources(\_playerA, resourcesGiven, resourcesWanted);

## yield return null;

## // Verificăm

## Assert.AreEqual(beforePlayerWood - 4, \_playerA.Resources[woodIdx]);

## Assert.AreEqual(beforePlayerBrick + 1, \_playerA.Resources[brickIdx]);

## Assert.AreEqual(beforeBankWood + 4, \_bankController.CurrentResources[woodIdx]);

## Assert.AreEqual(beforeBankBrick - 1, \_bankController.CurrentResources[brickIdx]);

## }

#### 2. BankTrade\_PartialBatch\_RemainderReturned

## [UnityTest]

## public IEnumerator BankTrade\_PartialBatch\_RemainderReturned()

## {

## int woodIdx = (int)ResourceType.Wood;

## int brickIdx = (int)ResourceType.Brick;

## // Setare bani inițiali

## int beforePlayerWood = \_playerA.Resources[woodIdx]; // ex. 99

## int beforePlayerBrick = \_playerA.Resources[brickIdx]; // ex. 99

## int beforeBankWood = \_bankController.CurrentResources[woodIdx];

## int beforeBankBrick = \_bankController.CurrentResources[brickIdx];

## // Jucător oferă 5 lumber, vrea 1 brick, fără buff adițional

## int[] resourcesGiven = new int[5];

## int[] resourcesWanted = new int[5];

## resourcesGiven[woodIdx] = 5;

## resourcesWanted[brickIdx] = 1;

## \_tradeController.TradeResources(\_playerA, resourcesGiven, resourcesWanted);

## yield return null;

## // 5 lumber → batch de 4, rămâne 1 în mână; primește 1 brick

## Assert.AreEqual(beforePlayerWood - 4, \_playerA.Resources[woodIdx]);

## Assert.AreEqual(beforePlayerBrick + 1, \_playerA.Resources[brickIdx]);

## Assert.AreEqual(beforeBankWood + 4, \_bankController.CurrentResources[woodIdx]);

## Assert.AreEqual(beforeBankBrick - 1, \_bankController.CurrentResources[brickIdx]);

## }

#### 3. BankTrade\_InsufficientBankResource\_DoesNothing

## [UnityTest]

## public IEnumerator BankTrade\_InsufficientBankResource\_DoesNothing()

## {

## int wheatIdx = (int)ResourceType.Wheat;

## int sheepIdx = (int)ResourceType.Sheep;

## // Jucător primește buff 2:1 la wheat

## \_playerA.PlayerBuffs.AddBuff(ResourceType.Wheat, PlayerBuff.Trade2\_1);

## \_bankController.CurrentResources[sheepIdx] = 0; // banca nu are oi

## int beforePlayerWheat = \_playerA.Resources[wheatIdx]; // ex. 99

## int beforePlayerSheep = \_playerA.Resources[sheepIdx]; // ex. 99

## int beforeBankSheep = \_bankController.CurrentResources[sheepIdx]; // 0

## int[] resourcesGiven = new int[5];

## int[] resourcesWanted = new int[5];

## resourcesGiven[wheatIdx] = 2;

## resourcesWanted[sheepIdx] = 1;

## \_tradeController.TradeResources(\_playerA, resourcesGiven, resourcesWanted);

## yield return null;

## // Niciun schimb:

## Assert.AreEqual(beforePlayerWheat, \_playerA.Resources[wheatIdx]);

## Assert.AreEqual(beforePlayerSheep, \_playerA.Resources[sheepIdx]);

## Assert.AreEqual(beforeBankSheep, \_bankController.CurrentResources[sheepIdx]);

## }

#### 4. BankTrade\_PlayerLacksResources\_DoesNothing

## [UnityTest]

## public IEnumerator BankTrade\_PlayerLacksResources\_DoesNothing()

## {

## int wheatIdx = (int)ResourceType.Wheat;

## int sheepIdx = (int)ResourceType.Sheep;

## // Jucător are doar 1 wheat, dar încearcă să dea 2

## \_playerA.Resources[wheatIdx] = 1;

## \_playerA.PlayerBuffs.AddBuff(ResourceType.Wheat, PlayerBuff.Trade2\_1);

## \_bankController.CurrentResources[sheepIdx] = 5;

## int beforePlayerWheat = \_playerA.Resources[wheatIdx]; // 1

## int beforePlayerSheep = \_playerA.Resources[sheepIdx]; // 99

## int beforeBankSheep = \_bankController.CurrentResources[sheepIdx]; // 5

## int[] resourcesGiven = new int[5];

## int[] resourcesWanted = new int[5];

## resourcesGiven[wheatIdx] = 2;

## resourcesWanted[sheepIdx] = 1;

## \_tradeController.TradeResources(\_playerA, resourcesGiven, resourcesWanted);

## yield return null;

## Assert.AreEqual(beforePlayerWheat, \_playerA.Resources[wheatIdx]);

## Assert.AreEqual(beforePlayerSheep, \_playerA.Resources[sheepIdx]);

## Assert.AreEqual(beforeBankSheep, \_bankController.CurrentResources[sheepIdx]);

## }