Deeluitwerking 2

Het maken van een programma van ons bordspel

Toen we de regels en design van ons spel hadden gemaakt werd het tijd om deze te digitaliseren. Dit kan op meerdere manieren maar we kozen eigenlijk gelijk al voor Unity, een programma waar je games in kunt maken, omdat we hier al ervaring mee hadden en het een krachtig programma is met een groot gebruikers aantal. Dit betekende dat we de basis niet meer hoefde te leren en als we een probleem tegenkwamen, het antwoord meestal wel op het internet staat. Het proces van het maken van het spel bestaat uit een aantal stappen, namelijk:

- Creëren van de grafische benodigdheden
- Stukken Laten bewegen en slaan
- Winnen mogelijk maken
- UI maken en laten functioneren

Creëren van de grafische elementen

Om te beginnen heb je natuurlijk plaatjes nodig die je uiteindelijk kunt laten bewegen. Hiervoor hebben we een paar eisen opgesteld waaraan voldaan moest worden, deze staan hieronder.

- Het spel gebruikt zachte kleuren voor het bord en UI zodat het makkelijk op het oog is en gebruiksvriendelijk
- De stukken hebben een minimalistisch ontwerp waar makkelijk duidelijk uit gemaakt kan worden welke zetten deze mogen spelen

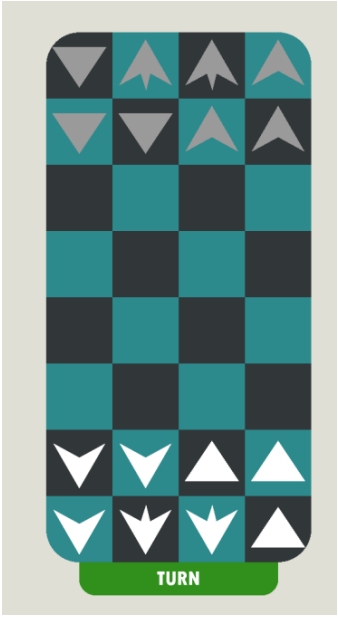
Kleuren

Naar aanleiding van de eisen hebben we de stukken verder ontworpen. Voor de kleuren zijn we naar een kleurenpalet website (https://coolors.co/) gegaan en hebben daar net zolang kleuren gegenereerd totdat we op een goed palet uitkwamen. Uiteindelijk zijn we gekomen op een zeegroen en donkerblauwige-grijs voor de vlakken van het bord. Dit zijn goed contrasterende kleuren en beide dof en dus licht op de ogen. Voor de achtergrond hebben we een beige gepakt omdat contrasteert met de kleuren die we hadden genomen voor de vlakken en ook dof was en dus niet vermoeiend om naar te kijken. Voor de zet aangevers hebben we een groen gepakt voor normale zetten en een rood voor zetten waarbij je een ander stuk slaat. Dit leken ons goede kleuren aangezien deze gelijk hun functie aangaven, namelijk groen voor goede zet en rood voor agressieve/aanvallende zet, en ook weer zacht en contrasterend waren. Voor de kleuren voor de Stukken hebben we gekozen voor de klassieke zwart en wit omdat dit een veelgebruikt palet is en deze ook wel bij onze kleuren paste. Het enige probleem wel is dat puur zwart een slecht contrast heeft met onze donkergrijs, zoals hiernaast zichtbaar, en dus hebben we zwart verander in grijs.

Ontwerp

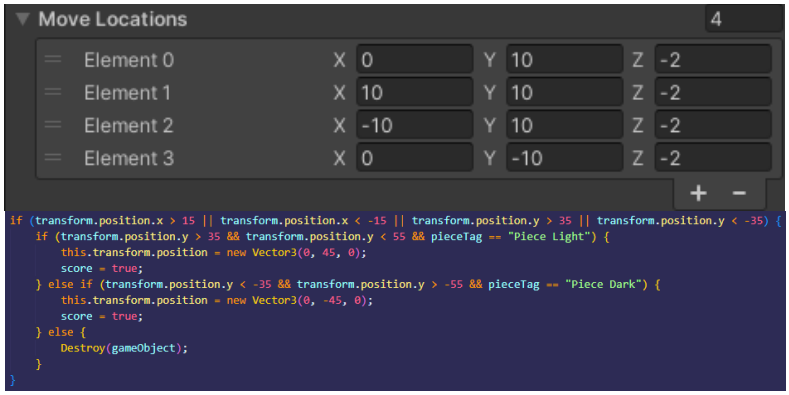
Nadat we de kleuren hadden bepaald gingen we door naar het ontwerp van de stukken. Volgens onze eisen moesten deze aangeven welke zetten deze kunnen spelen dus dat was een belangrijke factor in het ontwerpen van de stukken. We kwamen er dan ook snel achter dat simpele vormen waarbij de punten verwijzen naar de zetten die dat stuk mag spelen het beste overeenkwam met onze eisen. Deze stukken hebben we daarna gemaakt in Gimp, een gratis foto-bewerking programma dat lijkt op Photoshop, van driehoeken. Zoals op de screenshots pagina zichtbaar bestaat bijvoorbeeld het nu θ (Théta) stuk uit drie identieke uitgerekte driehoeken. Voor

het bord wouden we het een soepel ontwerp met ronde randen. Deze ronde randen hebben we ook in Gimp gemaakt doormiddel van een cirkel die alle zijkanten van het plaatje aanraakt te maken en dan de overgebleven stukjes in te kleuren behalve 1. Om aan te geven waar de stukken heen kunnen bewegen hebben we gekozen voor cirkels die dan over de mogelijke vlakjes komen. Deze zijn gemaakt door een cirkel te maken en daar dan een cirkel uit te halen. Verder hebben we nog een soort groene balk gemaakt omdat we tijdens het spelen het af en toe lastig vonden om te onthouden wie er aan de beurt was. Deze wisselt steeds naar de kant van de speler die aan de beurt is. Deze hebben we op dezelfde manier gemaakt als de hoekstukken maar dan twee keer met een balk ertussen. Als je dit dan allemaal samenvoegt dan krijg je het resultaat dat nu te zien is in de game en hier rechts.



Stukken laten bewegen

Nadat we een ontwerp voor de stukken hadden en deze hadden geïmporteerd in Unity was het tijd om het spel functioneel te maken. De stukken moesten bewegen volgens de regels van het stuk en mochten niet op elkaar landen, behalve bij de stukken van een andere kleur maar die moesten dan ook weer gelagen worden. Eerst moesten we zorgen dat stukken geselecteerd konden worden. Dit hebben we gedaan door een script (S5) te schrijven waar we de positie van de muis konden krijgen als je op de linkermuisknop drukt. Hiermee schieten we op die positie een raycast af, dit kun je zien als een soort laser die oneindig dun is, en gaan we kijken of die laser een stuk raakt. Als de raycast dan inderdaad een stuk raakt dan sturen we een signaal naar dat stuk zodat hij weet dat geselecteerd is. Als een stuk dan geselecteerd wordt, gaat hij kijken waar hij allemaal heen zou kunnen bewegen. Deze mogelijke posities worden berekend door de mogelijke verschuivingen op te tellen bij de positie waar het stuk nu staat. Deze verschuivingen hebben we van tevoren in een lijst gezet en deze aan het stuk script van het stuk gegeven. De stukken zijn in Unity 10 bij 10 groot dus verplaatsen ze ook per 10.




```
if (transform.position.x > 15 || transform.position.x < -15 || transform.position.y > 35 || transform.position.y < -35) {
    if (transform.position.y > 35 && transform.position.y < 55 && pieceTag == "Piece Light") {
        this.transform.position = new Vector3(0, 45, 0);
        score = true;
    } else if (transform.position.y < -35 && transform.position.y > -55 && pieceTag == "Piece Dark") {
        this.transform.position = new Vector3(0, -45, 0);
        score = true;
    } else {
        Destroy(gameObject);
    }
}
```

Hier rechts is een voorbeeld van de zetten van ons θ Théta stuk. Voor elke nieuwe positie die hier uitkomt wordt eerst een check gedaan om te kijken of de zet legaal is of niet. Er wordt dus gekeken of de nieuwe zet wel op het bord zit en of deze niet op een ander stuk landt. Verder wordt er gekeken of deze zet een stuk van het bord af speel zoals het bedoeld is en als dat gebeurt wordt de cirkel naar het midden verplaats op de x-as om duidelijk aan te geven dat het een speciale zet is die belangrijk is om te spelen. Als een zet al deze checks heeft doorstaan wordt hij weergegeven aan de speler doormiddel van de groene cirkel voor normale zetten en een rode cirkel voor zetten waarbij je een ander stuk slaat. Als deze cirkel dan wordt aangeklikt dan laat die aan het stuk weten dat die daarheen verplaatst kan worden haalt het stuk als die verplaatst al de groene en rode cirkels weer weg. Als een stuk wordt geslagen wordt die ook van het bord gehaald. Er wordt ook bijgehouden hoeveel stukken er per soort en in totaal worden geslagen en daar wordt meer uitleg over gegeven in de volgende onderwerpen. Als laatste wordt de beurd omgedraait en is de andere speler aan de beurt. De groene balk die de beurt aangeeft verdwijnt hierbij ook achter het bord en degene van de andere speler komt tevoorschijn.

Winnen mogelijk maken

Winst detecteren

Een belangrijk deel van een spel is de mogelijkheid om te winnen en die kon dus ook in ons spel niet ontbreken. Hiervoor moesten we dus detecteren wanneer iemand gewonnen had. In ons spel zijn er maar 2 mogelijkheden om te winnen dus het viel redelijk mee om dit onderdeel werkend te krijgen. We hoefden namelijk alleen maar te detecteren wanneer iemand een stuk van het bord af speelt of wanneer een speler geen stukken meer over heeft. De tweede is technisch gezien overbodig maar versnelt het proces van winnen wel enorm. De eerste is gedaan door bij het script dat kijkt of een mogelijke zet een winnende zet is toe te voegen dat als deze dan ook gespeeld wordt, hij aangeeft aan het overkoepelende script dat de speler die deze gespeeld heeft, heeft gewonnen. De tweede is gedaan door een variabele te maken die bijhoudt hoeveel stukken er van een kleur over zijn. Deze begint altijd op 8 aangezien er acht stukken per kleur zijn en elke keer dat een stuk geslagen wordt, word er 1 afgetrokken. De variabele wordt constant in de gate gehouden door het hoofd script en als deze dan op 0 terecht komt wordt het winproces in actie gezet.

Winnen

Als een speler heeft gewonnen moet dat ook duidelijk worden gemaakt voor de speler dus we hebben een paar animaties toegevoegd. Alle stukken, inclusief het stuk dat van het bord af is gespeeld, worden verwijderd doormiddel van een krimp animatie. Hierdoor is het bord helemaal leeg en kunnen we er andere dingen op zetten. De krimp animatie is bedoeld zodat het wat duidelijker wordt. Dan halen we ook de beurt indicator weg, aangezien die niet meer van toepassing is. Als laatste plaatsen we tekst op het veld zodat je kan lezen wie er gewonnen heeft voor minimale verwarring. Dit helpt ook bijvoorbeeld bij een potje waar de AI tegen zichzelf speelt aangezien dat best snel kan gaan. Uiteindelijk komt het er dan uit te zien zoals het hier rechts ook staat. In dat voorbeeld heeft win gewonnen maar dat zou uit het plaatje dus duidelijk moeten worden.

UI maken en laten functioneren

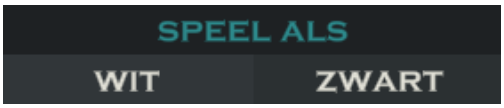
Als laatste kwam het maken van de UI om het spel heen. Het spel werkt in zijn volledigheid maar als gebruiker is het toch prettig om een aantal extra dingen te weten. We hadden een paar dingen die we graag wouden toevoegen en dat waren:

- Menu om modus en dergelijke te selecteren
- Geslagen stukken per kleur
- Legale zetten en spelregels

Menu

Allereerst wouden wij graag een menu waarin je de belangrijke dingen van het spel kon selecteren. Hierin moest je kunnen kiezen of je tegen de AI wou spelen, tegen een ander persoon of gewoon wil toekijken hoe de AI het met zichzelf uitvecht. Als je tegen de AI speelt moet je kunnen bepalen als welke kleur je wilt spelen en je moet het bord kunnen resetten als je opnieuw wilt beginnen. Ook moet je de spelregels tevoorschijn kunnen toveren en moet je het spel kunnen afsluiten. De titel van het spel erbij zetten vonden wij ook goed passen bij het spel en zorgt ervoor dat als iemand aan het spelen is, wij gratis reclame hebben. En als laatste vonden we het belangrijk dat duidelijk was wie het spel heeft gemaakt dus wouden we onze namen er ook in hebben staan. Dit betekende dat we ook in de code zouden moeten zorgen dat we konden wisselen van modus en menu's konden laten opdoemen. Het wisselen van de modus hebben we gedaan door een variabele aan te maken die Unity onthoudt, hoe vaak je hem ook opnieuw opstart, en daar de modus in op te slaan. Dit betekent dat we het spel kunnen herstarten en nog steeds van modus wisselen. Daarna kijkt het hoofdscript welke modus geselecteerd is en stuurt de AI daarop aan. Bij de modus speler tegen AI wacht het spel

even met herladen en laat het eerst een menuutje tevoorschijn komen waarin de speler kan kiezen als welke kleur hij speelt, en herlaadt hij het spel nadat de speler zijn keuze heeft gemaakt.



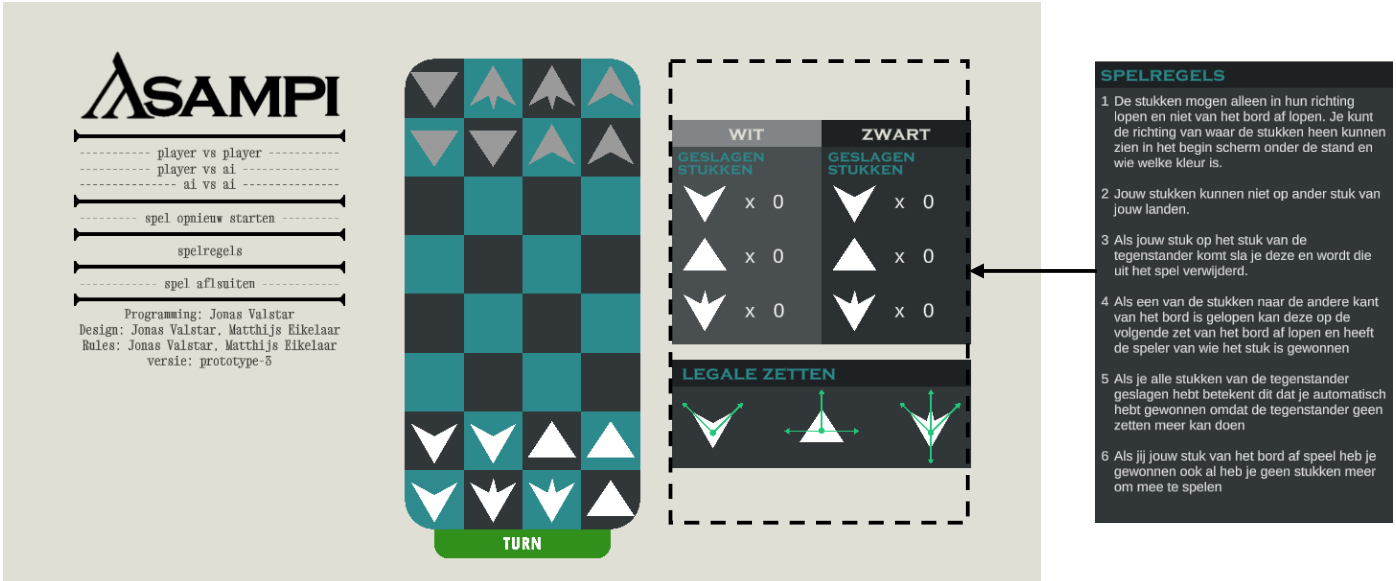
Voor de herstart functie herlaadt het spel gewoon zodat de stukken weer naar de begin positie teruggaan. Aangezien het spel dus kan onthouden in welke modus hij zit na het herstarten hoeft de speler dus niet opnieuw te selecteren welke modus hij wil spelen. Voor de spelregels hebben we simpelweg een tekst vak gemaakt met de regels erin en een achtergrond. Deze staat standaard op non-actief waardoor hij niet zichtbaar is, maar als de speler op de spelregels knop drukt wordt deze door het script op actief gezet en kan de speler de spelregels lezen. Het spel afsluiten is erg simpel en is gewoon een functie in Unity.

Geslagen stukken

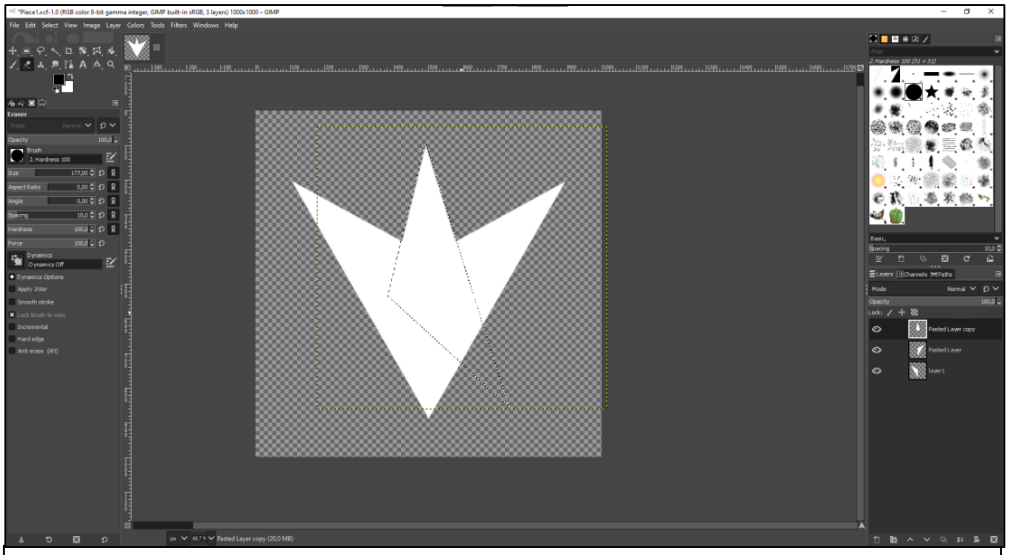
Voor schaakspelers is het essentieel om te kunnen zien welke stukken ze hebben geslagen om te kijken hoe goed ze ervoor staan. Wij moesten dit dus ook in ons spel implementeren aangezien ons spel wel wat weg heeft van schaken en Matthijs een schaakliefhebber is. Dit was niet heel lastig te implementeren. We hebben een stuk aan de rechterkant van het bord gepakt en hebben daar een donkergrijs vlak overheen gezet. Deze hebben we onderverdeeld in een kant voor de witte speler en de zwarte speler. Hier hebben we de drie verschillende stukken in geplaatst met een nummer erachter om aan te geven hoeveel van dat stuk de desbetreffende speler heeft geslagen. Dit was makkelijk te implementeren aangezien we al een functie hadden voor het slaan van de stukken en 1 van het totaal af halen. We hoefden hier alleen maar een stukje toe te voegen dat keek van welk type het geslagen stuk was en dan 1 bij dat type op te tellen.

Zetten en spelregels

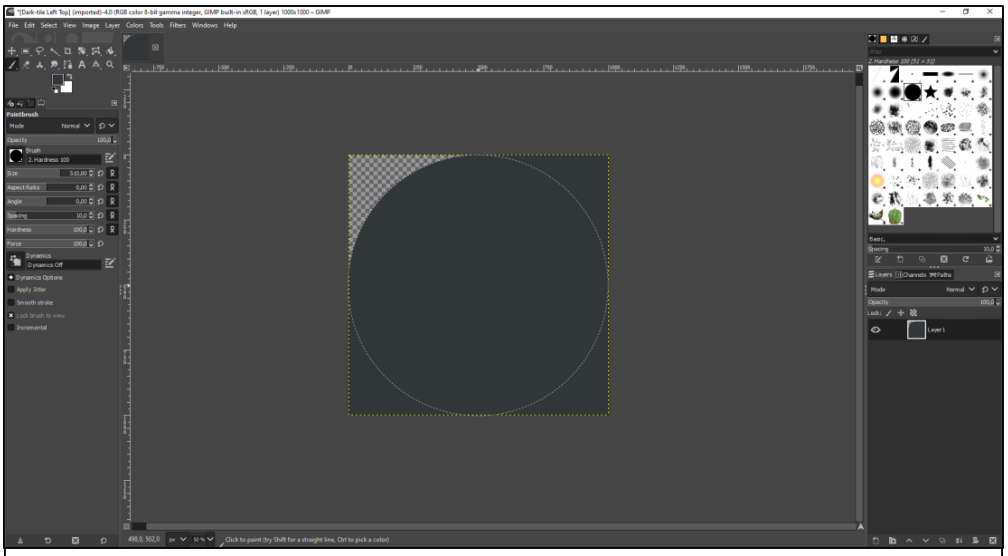
Dit was misschien wel het makkelijkste van het hele digitaliseringsproces. We hebben in Gimp gewoon de stukken gepakt en er pijltjes overheen getekend in de richting van de zetten die het stuk mag doen. Deze hebben we een achtergrond gegeven en onder de geslagen stukken gezet. De spelregels hebben we op dezelfde manier gemaakt alleen dan met tekst in plaats van plaatjes. De enige moeilijkheid daar was dat het er mooier uitzag als de regeltekst allemaal rechts van het regel nummer bleef staan dus die zijn onderverdeeld in 2 tekstvakken. Het spelregelvlak gebruikt dezelfde techniek als de kleurkeuze die eerder is uitgelegd en is dus niet zichtbaar als je het spel opstart. Het eindresultaat is hieronder te zien waar de stippellijn het stuk aanduidt waar de spelregels staan als die geselecteerd is.



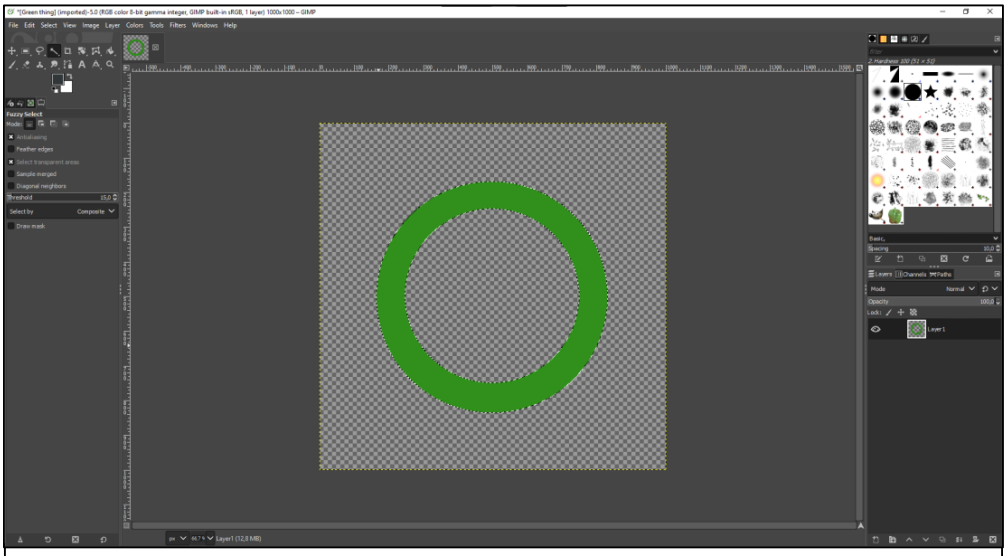
Screenshots



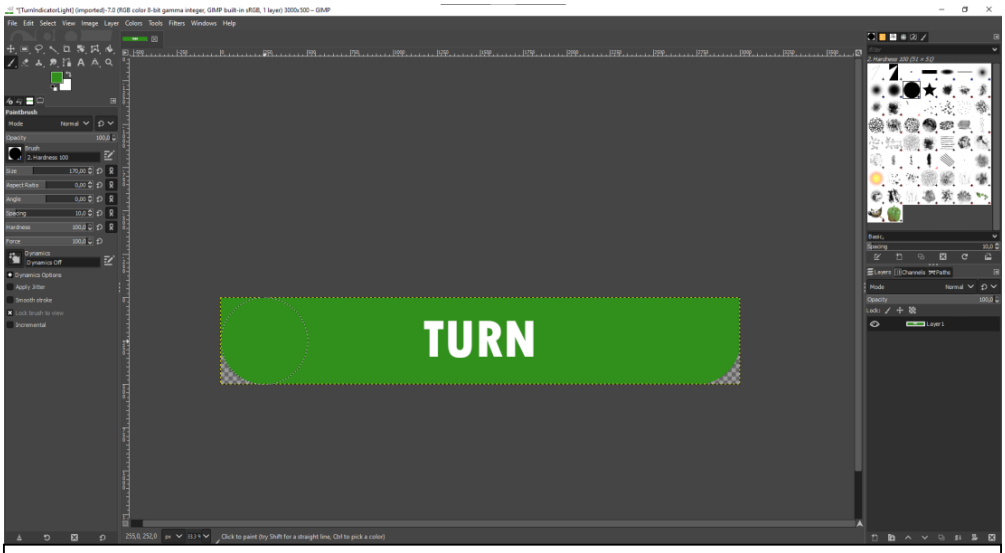
S1: Théta stuk in de maak in Gimp en gemaakt van 3 identieke uitgerekte driehoeken



S2: Hoeken van het bord zijn simpele cirkel met 3 reststukje ingekleurd



S3: Zet aangeef stukjes zijn cirkel met een andere cirkel eruit gehaald.



S4: Turn balk is gemaakt met dezelfde methode als de hoekstukken

```
if (Input.GetMouseButtonDown(0)) {
    Vector2 raycastPosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
    RaycastHit2D hit = Physics2D.Raycast(raycastPosition, Vector2.zero);

    if (hit.collider != null) {
        if (hit.collider.tag == "Piece Light" || hit.collider.tag == "Piece Dark") {
            if ((hit.collider.tag == "Piece Light" && turn == 1) || (hit.collider.tag == "Piece Dark" && turn == -1)) {
                if (hit.collider.gameObject.GetComponent<Piece>().isSelected != true) {
                    hit.collider.gameObject.GetComponent<Piece>().isSelected = true;
                } else {
                    hit.collider.gameObject.GetComponent<Piece>().isSelected = false;
                }
            }
        } else if (hit.collider.tag == "Movement Thing") {
            hit.collider.gameObject.GetComponent<Mover>().isSelected = true;
            ChangeTurn();
        }
    }
}
```

S5: script om

```
if (isSelected == true) {
    if (transform.childCount == 0) { // check for already spawned circles

        // unselecting other pieces
        if (selection.somethingSelected == true) { // checking is something is selected
            selection.selectedTransform.GetComponent<Piece>().isSelected = false; // making piece unselected, stopping spawning of circles
            foreach (Transform child in selection.selectedTransform) {
                GameObject.Destroy(child.gameObject); // delete movement circles
            }
        } else { // letting the main script know something has been selected
            selection.somethingSelected = true;
        }

        // making piece selected
        selection.selectedTransform = transform;

        // spawn circles
        foreach(Vector3 location in pieceSO.MoveLocations) {
            // checking occupancy of tiles
            RaycastHit2D hit = Physics2D.Raycast(transform.position + location, Vector2.zero);
            if (hit.collider != null && hit.collider.tag != transform.tag) { // spawns RED circle if piece with different tag already on target tile
                GameObject redCircle = Instantiate(circleEnemy, transform.position + location, Quaternion.identity, this.gameObject.transform);
                redCircle.GetComponent<Mover>().toBeDeletedPiece = hit.collider.gameObject;
                redCircle.GetComponent<Mover>().pieceTag = transform.tag.ToString();
                redCircle.name = "Deletion";
            } else if (hit.collider == null) { // creates a GREEN circle is tile is empty
                GameObject greenCircle = Instantiate(circle, transform.position + location, Quaternion.identity, this.gameObject.transform);
                greenCircle.GetComponent<Mover>().pieceTag = transform.tag.ToString();
            } // does not spawn a circle if piece of same tag already on target tile
        }
    }
}
```

S6: script dat de cirkels plaatst en kijkt of je een stuk kan slaan en of je op een ander stuk beland

```
if (isSelected == true) {
    if (score == true) {
        Vector3 movePosition = new Vector3(transform.position.x, transform.position.y, transform.position.z);
        pieceScript = GetComponentInParent<Piece>();
        pieceScript.isSelected = false;
        pieceScript.ScorePiece(movePosition);
        isSelected = false;
    } else {
        Vector3 movePosition = new Vector3(transform.position.x, transform.position.y, transform.position.z);
        pieceScript = GetComponentInParent<Piece>();
        pieceScript.isSelected = false;
        if (name == "Deletion") {
            if (toBeDeletedPiece.transform.tag == "Piece Light") {
                Debug.Log("1" + toBeDeletedPiece.gameObject.GetComponent<Piece>().type);
                switch(toBeDeletedPiece.gameObject.GetComponent<Piece>().type) {
                    case 0:
                        selection.lightZeta++;
                        break;
                    case 1:
                        selection.lightEta++;
                        break;
                    case 2:
                        selection.lightTheta++;
                        break;
                }
                selection.totalLight -= 1;
            } else {
                Debug.Log("2" + toBeDeletedPiece.gameObject.GetComponent<Piece>().type);
                switch(toBeDeletedPiece.gameObject.GetComponent<Piece>().type) {
                    case 3:
                        selection.darkZeta++;
                        break;
                    case 4:
                        selection.darkEta++;
                        break;
                    case 5:
                        selection.darkTheta++;
                        break;
                }
                selection.totalDark -= 1;
            }
            selection.UpdateCaptureTexts();
            Destroy(toBeDeletedPiece);
        }
        pieceScript.MovePiece(movePosition);
        isSelected = false;
    }
}
```

S7: script dat het stuk verplaats en andere stukken eventueel slaat

```
public void changePlayMode(int mode)
{
    PlayerPrefs.SetInt("playMode", mode);
    if (mode != 2) {
        Restart();
    }
}
```

S8: script dat zorgt voor het wisselen van beurt

```
public void OpenChooseMenu(bool open) {
    chooseMenu.SetActive(open);
    if (open == false) {
        Restart();
    }
}
```

S8: script dat zorgt voor het openen van het kleurmenu en als het sluit opnieuw laden van het spel