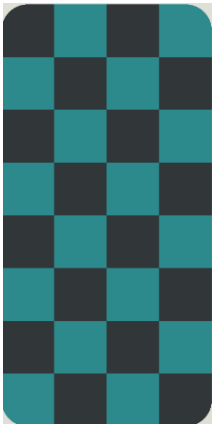
Ontwerp

Deeluitwerking 1

Het ontwerpen van ons eigen bordspel

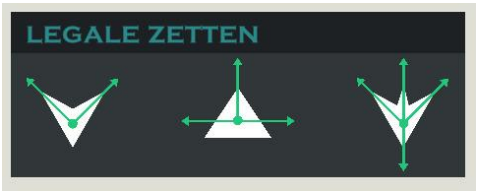


Voor het eerste deel van ons ontwerp zijn we begonnen met eisen stellen voor ons bordspel.
Onze eisen waren:

1. Basisregels mogen niet langer dan 2 minuten kosten om te begrijpen
2. Spel mag gemiddeld niet langer dan 8 minuten duren met een gemiddeld speeltempo van een zet per 10 seconden (48 zetten)
3. Het spel heeft weinig speciale zetten waardoor je sneller een betere zet kan spelen
4. Het spel heeft weinig speciale regels en een overzichtelijk klein bord waardoor er geen verwarring ontstaat met betrekking tot de mogelijke zetten van jezelf en van de tegenstander

Ontwerp

Op deze manier zijn we op ons spel gekomen; We zijn als eerste begonnen met het maken van een speelveld. We hebben gekozen voor iets wat lijkt op schaken en dammen, omdat dit makkelijk te begrijpen is voor anderen. Toen zijn we bezig geweest met de lengte en breedte van het bord. We hebben gekozen 8 bij 4 zodat je nog wel wat ruimte in de lengte hebt, maar niet te veel in de breedte. Hierdoor wordt het spel sneller, omdat er minder mogelijkheden zijn. Wij wilden een spel maken waar je naar de overkant moet lopen en elkaar van het bord af moet kunnen slaan. Dit is ook nog een reden waarom het bord langer is dan breed. Hierdoor moet je wel even doen om aan de overkant te komen. Ook kan je met een minder breed bord niet zo makkelijk om de tegenstander heen. Toen zijn we bezig gegaan met de stukken. Eerst hebben we de moves van onze stukken bepaald. Alles moest zo veel mogelijk naar voren zodat je niet te veel geschuif krijgt. Toen zijn we op deze vormen uit gekomen, want we hadden besloten dat de punten moesten wijzen naar de kanten waar dit stuk heen moest kunnen bewegen. Bij een van onze stukken ging dit niet helemaal omdat als wij deze logica zouden gebruiken, je een rechte lijn zou hebben en dat ziet er minder mooi uit. Ook hebben wij Matthijs zijn broertje Bram en onze vrienden de game meerdere keren laten testen zodat wij er van konden leren. Met hun feedback konden wij dan weer aanpassingen maken aan het design van het spel.



Stappenplan

In onze theorie staat een stappen plan voor het maken van ons bordspel, deze hebben we dan ook gebruikt. We zijn bij de eerste stap begonnen op de tweede manier. Dat is dat je begint met een spelconcept. We zijn begonnen met het maken van het bord. Ook hadden we al snel bepaald dat wij een spel op basis van tactiek wilden. Dit zodat het duidelijk zou zijn hoe lang een spelletje zou duren. Ook vonden wij het leuker dat je over een spelletje moet nadenken dan dat het op geluk gaat.

Stap 2

Toen zijn we bij vrienden en familie gaan testen of die er tevreden over waren. We moesten nog wel veel veranderen aan hoe onze game er uit zag. Het spel vonden ze goed omdat ze het snel konden spelen en het doel duidelijk was, maar ze wisten niet wat de stukken deden en konden dit ook niet opzoeken. Hier zijn we toen mee aan de slag gegaan.

Stap 3

Deze stap is vrij uitgebreid omdat je veel dingen bij langs moet gaan.

Gimmick: onze Gimmick is dat ons spel en stukken letters zijn vanuit het Griekse alfabet. Hierdoor valt het op en heeft het er een soort mysterie omheen.

Mogelijke manieren om te winnen: we hebben gekozen voor twee manier om te winnen. We wilden dat je kon winnen als je aan de overkant was. Ook hadden wij gezien dat het ook wel gebeurt dat een speler geen stukken meer had. Als dit gebeurde moest de ander naar de overkant lopen en won dan pas. Dit vonden we niet leuk, dus hebben we er toen voor gekozen om te doen dat als je alle stukken hebt, dat je dan ook wint, want je wint sowieso wel.

Interactie: er is best wel wat interactie als je wilt. Je kan namelijk met z'n tweeën spelen, omdat het spel niet te serieus is kan je gewoon gezellig praten.

Tot hoever in het spel is er nog een kans om te winnen: vooral als je nog niet zoveel speelt kan je nog steeds winnen. Ook is het een spel met veel tactiek waardoor je nooit hebt verloren tot het einde als de tegenstander maar slecht genoeg speelt kan je nog altijd winnen.

Spelers moeten het hele spel kunnen mee spelen: het is een één tegen één. Hierdoor moeten de twee spelers blijven spelen totdat het spel is afgelopen.

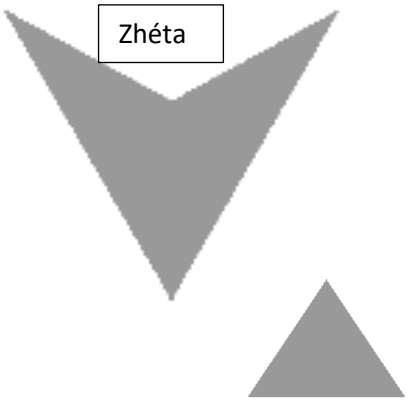
Race-element: ons spel is een grote race en dat is dan ook ons race-element.

Is het doel duidelijk: het doel is duidelijk winnen. Ook kunnen de spelers er makkelijk achter komen hoe, dat staat in de spelregels. Als je het zou vergeten kan je altijd weer even kijken en door spelen.

stukken

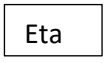
ζ Zheta

De Zheta beweegt altijd diagonaal naar voren. Dit stuk is gemaakt om mee aan te vallen. Het stuk kan niet naar achteren of opzij. Hierdoor staat het stuk snel voor aan in de strijd en kan het de verdedigende 'Eta' makkelijk omzeilen. Het ontwerp van het stuk is zo gemaakt dat de punten van het stuk de mogelijke beweegrichtingen aangeven. De tijdelijke naam van dit stuk was dan ook aanvaller, omdat dat het enige doel van dit stuk is. Dit stuk is ook deels geïnspireerd door de pion van het spel schaken met als uitzondering dat hij ook diagonaal beweegt in plaats van alleen slaat.



η Eta

De Eta beweegt naar links, naar rechts of naar voren. Het doel van dit stuk is om andere stukken tegen te houden zodat ze niet naar de overkant kunnen. Dit stuk kan niet achteru it of diagonaal. Dit maakt dit stuk een slechte aanvaller aangezien hij maar 1 vlak aanvalt maar uitermate geschikt om als verdedegende muur te fungeren.Het ontwerp van dit s tuk is gemaakt zodat de punten naar de beweegrichtingen wijzen en de vlakke stukken geven ook de zwakste plekken aan van dit stuk en de locatie waarvan je wilt aanvallen.

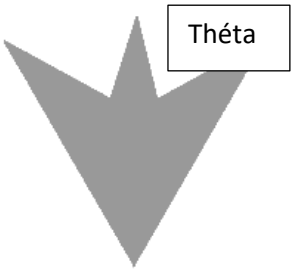


θ Theta

De Theta beweegt diagonaal naar voren en kan ook nog recht naar voren en naar achteren. Dit is het enige stuk dat naar achteren kan bewegen, wat hem erg waardevol maakt. Het stuk is daarom goed te gebruiken als rugdekking voor de stukken in de frontlinie. Het is echter cruciaal dat hij snel naar achteren kan als er een doorbraak in de linie ontstaat en deze wil je dus meestal niet gebruiken als directe aanval. De punten van het ontwerp van dit stuk duiden, net als de andere bij de stukken,

op de bewegings richtingen van dit stuk. Ook is de vorm van een kroon met de drie punten terug te vinden, wat aangeeft dat dit het sterkste stuk uit het spel is.

Testen



We hebben onze vrienden getest, maar we hebben vooral bij de profielwerkstuk middag veel kunnen testen. Er kwamen namelijk veel mensen kijken en spelen. Deze mensen hebben we vaak een beetje uitleg geven over het spel. Het ontwerp en ook de spelregels. Hierna speelde ze vaak een spelletje tegen de AI of als er meerdere mensen waren speelden ze tegen elkaar. Hier uit hebben we kunnen halen dat het spel niet moeilijk was om te spelen. Het speel tempo lag misschien iets hoger dan dat we hadden gehoopt. Het is wel zo dat mensen voor het eerst speelden en dus eerst geen idee hadden. Als mensen verder kwamen in het spel gingen ze naar het verwachte speeltempo. Er was ook een man en die geen uitleg wilde. Die kwam gewoon even kijken. Toen het wat rustiger werd kwam hij terug omdat hij graag toch een keer een potje wilde spelen. Het lukt dus wel om mensen het spel te leren zonder dat wij ook maar iets zeggen. Hij won ook nog eens dus hier kunnen we uit halen dat alles duidelijk is overgekomen.

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

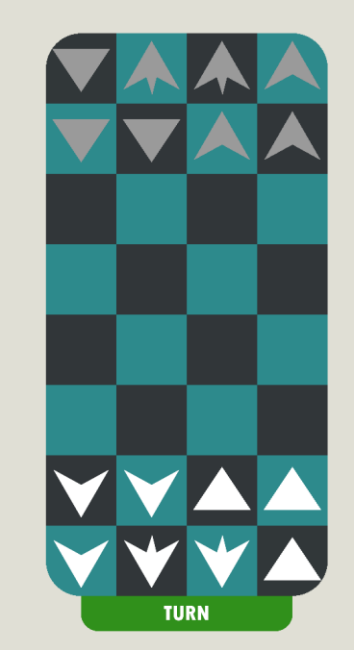
Kleuren

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

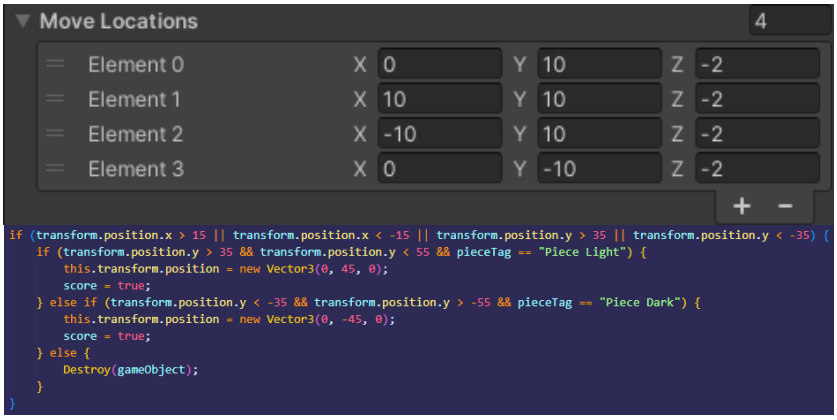
Nadat we de kleuren hadden bepaald gingen we door naar het ontwerp van de stukken. Volgens onze eisen moesten deze aangeven welke zetten deze kunnen spelen dus dat was een belangrijke factor in het ontwerpen van de stukken. We kwamen er dan ook snel achter dat simpele vormen waarbij de punten verwijzen 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.



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

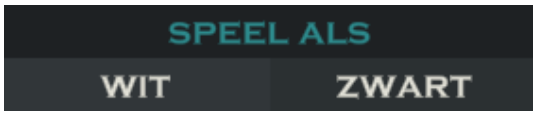
Als laatste kwam het maken van de UI om het spel heen. Het spel werkt in zijn volledigheid maar als gebruiker is het toch prettig om een aantal extra dingen te weten. We hadden een paar dingen die we graag wouden toevoegen en dat waren:

- 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.



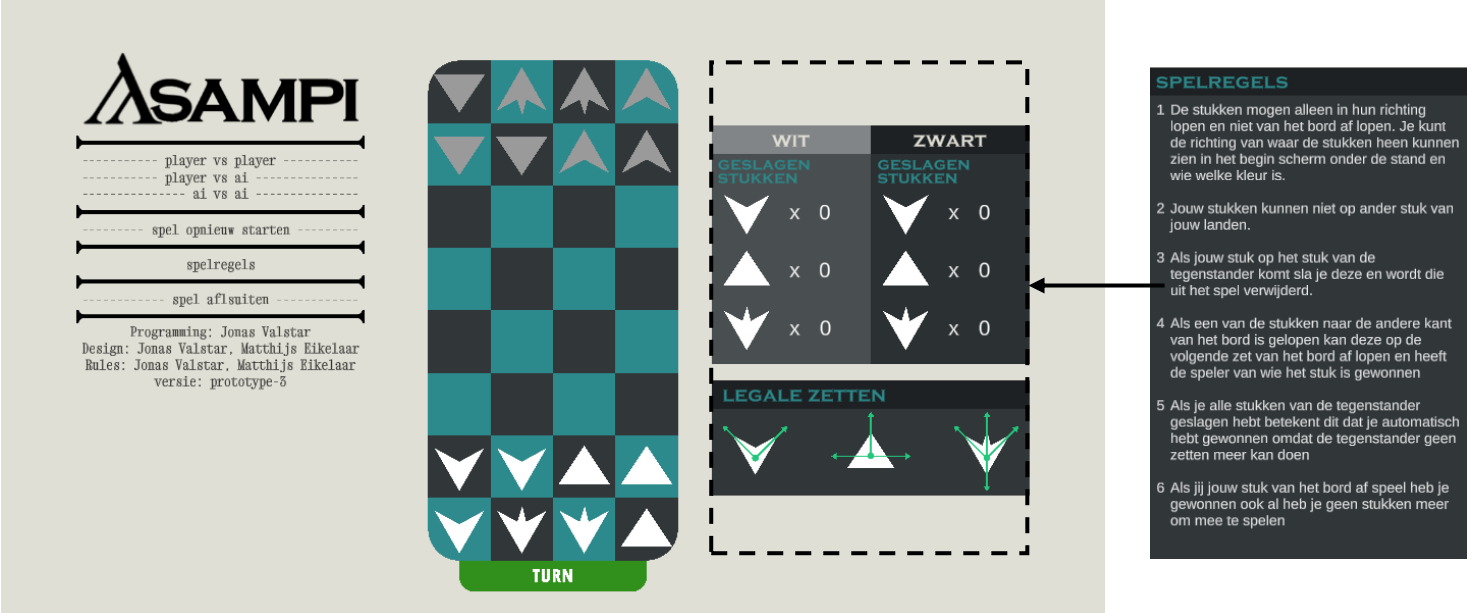
Voor de herstart functie herlaadt het spel gewoon zodat de stukken weer naar de begin positie teruggaan. Aangezien het spel dus kan onthouden in welke modus hij zit na het herstarten hoeft de speler dus niet opnieuw te selecteren welke modus hij wil spelen. Voor de spelregels hebben we simpelweg een 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

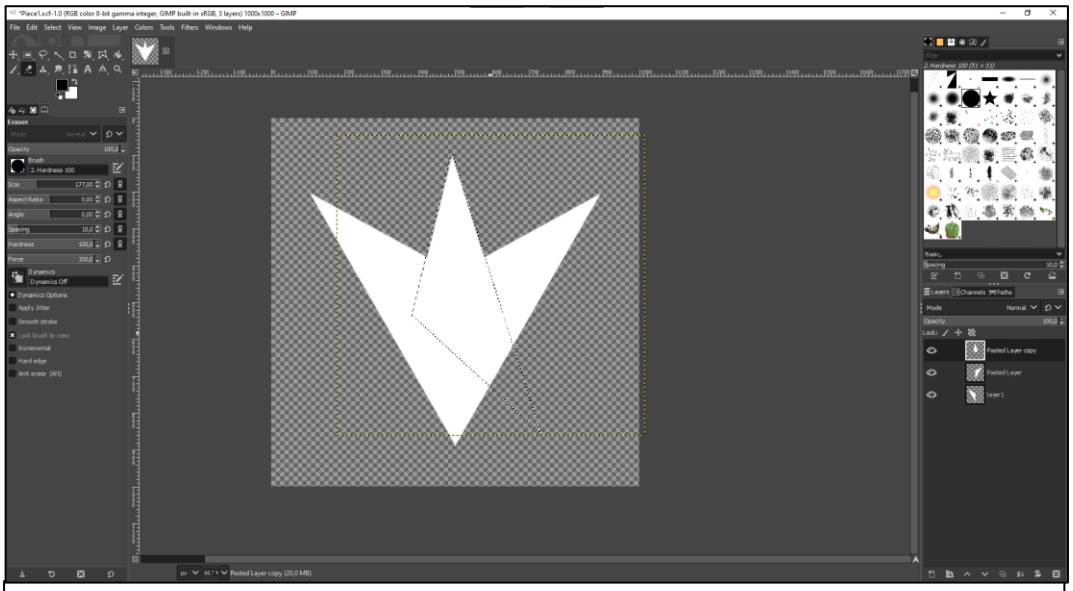
Voor schaakspelers is het essentieel om te kunnen zien welke stukken ze hebben geslagen om te kijken hoe goed ze ervoor staan. Wij moesten dit dus ook in ons spel implementeren aangezien ons spel wel wat weg heeft van schaken en Matthijs een schaakliefhebber is. Dit was niet heel lastig te implementeren. 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

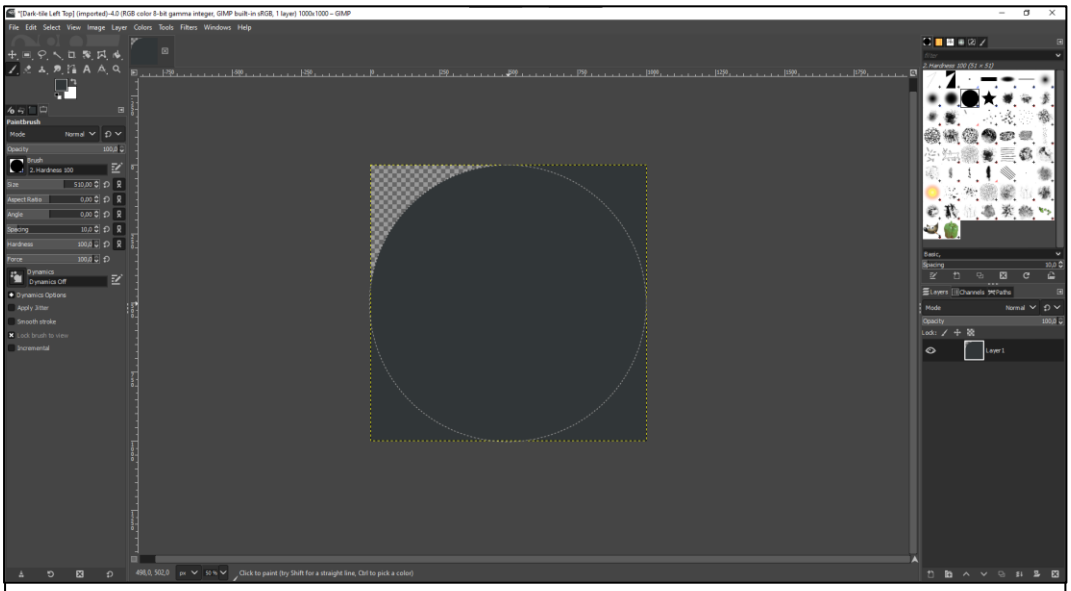
Dit was misschien wel het makkelijkste van het hele digitaliseringsproces. We hebben in Gimp gewoon de stukken gepakt en er pijltjes overheen getekend in de richting van de zetten die het stuk mag doen. Deze hebben we een achtergrond gegeven en onder de geslagen stukken gezet. De spelregels hebben we 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.



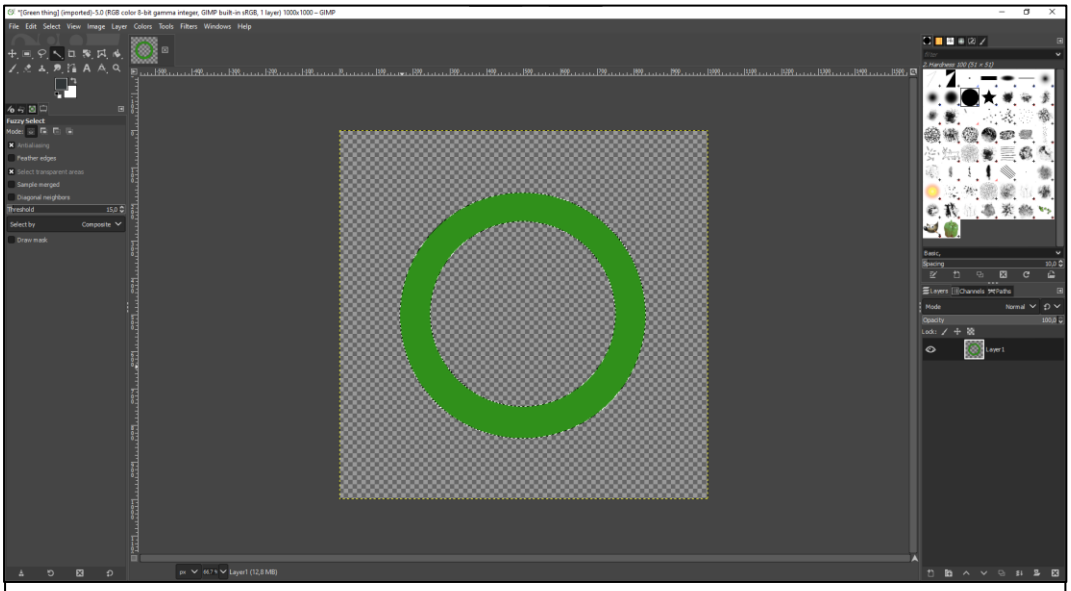
Screenshots



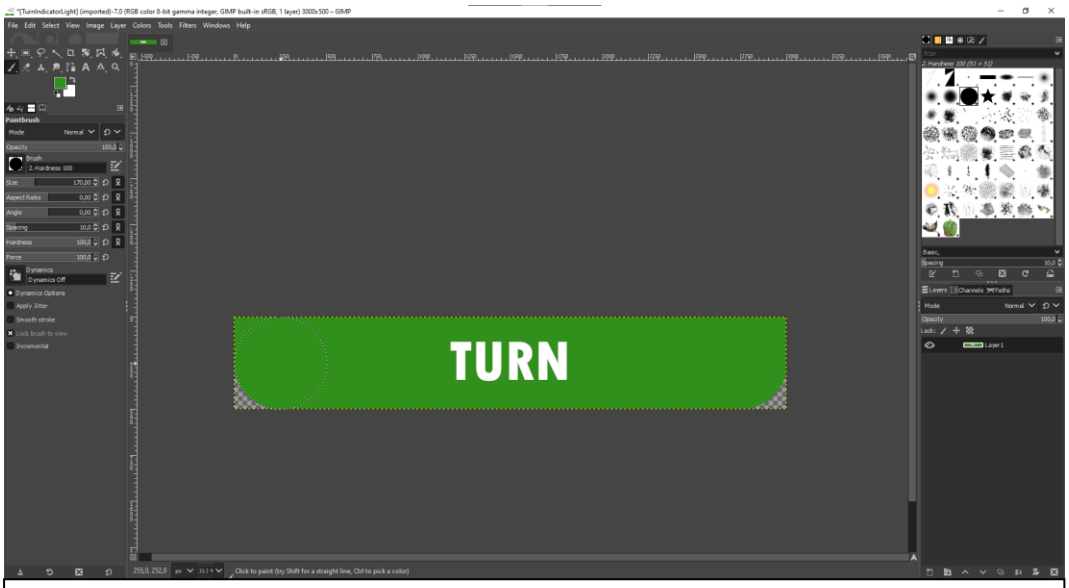
S1: Théta stuk in de maak in Gimp en gemaakt van 3 identieke uitgerekte driehoeken



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

Deeluitwerking 3

Het maken van een AI voor ons bordspel.

Het doel van het project was vanaf het begin al het maken van een AI, dus dit is een belangrijk onderdeel in het project. We hebben de AI direct in het spel zelf in Unity gemaakt waardoor we geen ingewikkelde dingen hoefden te doen met herkennen van stukken en posities en het verplaatsen daarvan, en is daardoor ook geschreven C#. Ons doel was om zelf een AI te maken en niet een van het internet te stelen. Dit zou je kunnen interpreteren door te denken dat we simpelweg niet letterlijk een AI's code mogen overnemen. Wij hebben het echter anders gedaan en hebben überhaupt niet naar de veelgebruikte strategieën gekeken. We zijn als het ware in het diepe gesprongen. Dit betekende dat we zelf een algoritme hebben ontwikkeld en zelf moesten uitzoeken hoe we de AI een AI gingen maken. Voor die laatste hebben we echter wel onderzoek gedaan omdat we hebben moesten weten wat nou precies een AI is. Ook hebben we gekeken hoe een AI leert en wat hiervoor nodig is. Een AI moet namelijk kunnen leren om als een AI geclassificeerd te worden. Verder was het handig om onderzoek te doen naar de geschiedenis van de AI om te kijken wat er allemaal al geprobeerd is en wat sowieso niet werkte. Hierbij hebben we ook gelijk gekeken hoe het werkt met copyright van AI's omdat dat misschien relevant zou zijn mochten we het spel en AI willen uitbrengen. We hebben ook goed gekeken naar AI's bij bordspellen, omdat dat ook is wat wij gaan doen. Hierbij rekening gehouden om niet al te veel naar de technische details te kijken omdat we dit zelf moesten doen. Met deze informatie was het dan eindelijk tijd om aan de slag te gaan. Voor onze AI moesten we een paar dingen doen. Hij moest namelijk:

- Het bord en de posities van stukken achterhalen
- Zetten maken en opslaan voor later.
- Proces herhalen voor een x-aantal zetten diep
- Berekenen welke zet het beste is
- Deze zet spelen
- Leren van zijn fouten

Posities achterhalen

Om een AI te later werken heb je natuurlijk gegevens nodig om erin te stoppen. In ons geval is dat hoe het bord erbij staat, en waar alle stukken staan. Met deze gegevens kan de AI daarna nadenken en berekeningen doen. In Unity heb je een ingebouwde functie dat je alle objecten met een bepaalt label kan vinden en in een lijst op kan slaan. Met deze kennis hebben we in Unity alle stukken een label gegeven met welke kleur dit stuk heeft. Deze kunnen we dan in het script opzoeken en in een lijst stoppen. Dit is alleen niet het enige wat we moeten doen. De AI moet als bijde kleuren kunen spelen, dus we moeten eerst kijken met welke kleur de AI eigenlijk speelt. Dit doen we door naar het Selectie script (het hoofd script) te gaan en op te vragen welke kleur aan de beurt is. Dan stoppen we kleur die aan de beurt is in het lijstje van de AI en de andere kleur in het lijstje van de tegenstander. Ook belangerijk is dat we opslaan van welk type een stuk is. Dit omdat we moeten weten welke zetten een stuk allemaal mag doen en we het niet kunnen opvragen bij de stukken zelf. De AI gaat namelijk een soort van in zijn hoofd alle zetten bij langs maar speelt deze niet. Als we dan tussen tijds gaan opvragen wat voor een stuk op een bepaalde positie staat, is op die plek misschien helemaal geen stuk te vinden of een verkeerd stuk.

Zetten maken en opslaan

Zodra we stukken hebben opgeslagen in het geheugen van de AI is het tijd om te kijken Welke zetten allemaal gespeeld kunnen worden. Om dit te doen heeft de AI natuurlijk de mogelijkheden nodig de gespeeld kunnen worden. Hiervoor hebben we een lijst gemaakt met alle stukken erin en hun mogelijke zetten. Als de AI dus bij de zetten wil gaat hij gewoon naar deze lijst, zoekt het stuk dat overeenkomt met het type dat hij opgeslagen heeft en leest de mogelijkheden af. Het zet zoek

```
// getting all first positions and moves
for (int i = 0; i < currentPositions.Count; i++) { // loops through every piece
    foreach(Vector3 move in piecePrefabs[currentPositionsType[i]].GetComponent<Piece>().pieceSO.MoveLocations) { // gets all possible moves a piece can make
        Vector3 newLocation = new Vector3(currentPositions[i].x + move.x, currentPositions[i].y + move.y, -1); // get the new location
        if ((newLocation.y > 40 && piecePrefabs[currentPositionsType[i]].tag == "Piece Light")) { // check to see if AI can win as Light
            gameObject.GetComponent<Selection>().WinGame(true); // winning game
            freePieceMove.Add(-1); // setting to -1 to stop AI from calculating further
        } else if ((newLocation.y < -40 && piecePrefabs[currentPositionsType[i]].tag == "Piece Dark")) { // check to see if AI can win as Dark
            gameObject.GetComponent<Selection>().WinGame(false); // winning game
            freePieceMove.Add(-1); // setting to -1 to stop AI from calculating further
        } else if(newLocation.x < 20 && newLocation.x > -20 && newLocation.y < 40 && newLocation.y > -40 && currentPositions.Contains(newLocation) == false) {
            // remember first moves
            FM_oldPosition.Add(currentPositions[i]);
            FM_newPosition.Add(newLocation);
            FM_type.Add(currentPositionsType[i]);
        }
    }
}
```

proces begint dan ook met voor elk stuk dat op het bord staat van de AI's kleur bekijken wat voor zetten die kan doen. Tijdens deze stap doet hij ook gelijk de nodige controles of het wel een legale zet is en kijkt hij of hij die zet kan winnen. Als het een zet is waardoor de AI onmiddellijk wint is dat natuurlijk sowieso de beste zet en wordt het hele vervolgproces onmiddellijk afgekapt en speelt de AI die zet. Als het geen winnende zet is en hij is legaal wordt deze toegevoegd aan een lijst met alle eerste zetten die mogelijk zijn, zodat de AI later kan achterhalen met welke zet de beste uiteindelijke positie mee begon. Hier wordt de oude positie, nieuwe positie en het type van het stuk opgeslagen zodat het spel exact weet welk stuk waarvandaan kwam en waar die naartoe moet als die zet uiteindelijk gespeeld wordt. Hierna worden de mogelijke zetten verwerkt door een ander stuk script. Dit stuk gaat langs alle mogelijke zetten stuk voor stuk en slaat ze op alsof hij die gespeeld heeft. Dit houdt in dat de AI het opgeslagen bord pakt en daar een kopie van maakt. Uit deze kopie verwijdert hij daarna de oude positie en voegt de nieuwe positie er terug in. Hij kijkt ook of er een stuk geslagen wordt en als dat zo is, dan wordt deze verwijdert uit de lijst van de AI's tegenstanders. Er wordt in dit stuk tekst ook nog wat meer controle gedaan voor zetten die de AI echt niet zou moeten spelen, Maar die worden later nog behandeld in het kopje 'Beste zet berekenen'.

Proces herhalen

Als de AI de eerste posities gevonden heeft is het natuurlijk zaak dat het proces herhaalt voor een x-aantal zetten. Een AI haalt namelijk zijn slimheid uit het ver kunnen vooruitdenken zodat het zetten van de tegenstander al aan kan zien komen en een plan kan bedenken die meerdere zetten duurt om te volbrengen. Dit werkt eigenlijk vrijwel hetzelfde als de eerste keer alleen moeten hier een paar stapjes extra doen om te zorgen dat het goed gaat. We hebben namelijk een aantal verschillende bord indelingen en deze gaan we stuk bij stuk bij langs en voeren hetzelfde proces op uit als de eerste keer. Een dingetje waar we hierbij wel rekening mee moeten houden is dat als de AI de nieuwe mogelijke indelingen bij de lijst toevoegd de lijst alleen maar langer wordt en de functie dus oneindig lang door kan gaan met berekenen omdat de lijst per 1 indeling uitgewerkt misschien wel 8 nieuwe erbij krijgt. Je krijgt dus een oneindige lus, en als er iets is wat computers niet leuk vinder is het een oneindige lus. We moeten dus nog een extra lijst aanmaken die als buffer dient en daar alles aan toevoegen. Als de AI dan alle originele indelingen heeft gehad, worden al die indelingen uit de lijst verwijderd en worden alle lijsten in de bufferlijst daarheen verplaatst. Zo kan het proces herhaald worden tot een bepaalde diepte en daar stoppen zonder dat het een oneinde lus wordt. Een ander ding is dat we natuurlijk wel om en om ook zetten voor de tegenstander moeten berekenenen. Dit werkt eigenlijk hetzelfde als voor de AI zelf alleen pakt het programma dan de stukken van de tegenstander en de mogelijke zetten van de tegenstander. Een derde ding waar we rekening meer moeten houden is dat we de uiteindelijk moeten kunnen traceren waar de beste indeling vandaan kwam. Hiervoor moeten we de eerste zet die gespeeld is onthouden. Dit doen we door de eerste zet per indeling op te slaan en mee tegeven. aan de functie. Deze functie pakt dan de eerste zet en zet die ook bij elke nieuwe indeling die uit de berekening rolt er weer bij.

Beste zet berekenen

Nadat de AI alle mogelijke eindindelingen heeft berekend is het tijd om te kijken welke eigenlijk het best is. Dit doet doormiddel van een paar dingen controleren en via een berekening te bepalen wat de waarde van een indeling is. De dingen die de AI controleert zijn:

- Totale Waarde van de stukken
- Gemiddelde positie van de stukken
- Hoe gedekt zijn stukken zijn.

Uit testen bleek dat dit alleen ervoor zorgde dat de AI af en toe ervan uitging dat de tegenstander hele domme zetten speelde, want dat zorgde ervoor dat hij er beter voor stond. Dit was natuurlijk niet de bedoeling, dus hebben we een paar beveiligingen geïmplementeerd waardoor de AI in ieder geval geen hele domme zetten zou spelen. Deze beveiligingen zijn:

- Geen stukken weggeven
- Gratis stukken altijd pakken.

Waarden van stukken

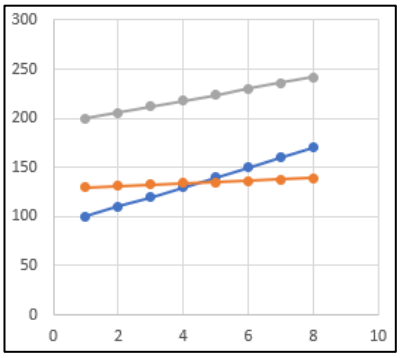
Hoeveel stukken je over hebt en hoeveel deze waard zijn is zo ongeveer het belangrijkste om te bekijken om te bepalen of een zet goed is of slecht. Dit houdt in dat de stukken een waarde moesten krijgen die representeerde hoe goed het stuk het stuk is voor de AI. Standaardwaarde is alleen niet het enige. Aangezien het in ons spel de bedoeling is om naar voren te gaan en door de verdediging van de tegenstander te breken, zijn stukken meer waard hoe verder ze onderweg zijn. Er zijn Echter stukken die je liever naar voren wilt hebben dan andere. Het ζ Zéta stuk bijvoorbeeld is bedoeld om aan te vallen dus die moet veel in waarde stijgen naarmate die verder komt. Het η Éta stuk is bedoeld als verdediger en moet dus juist weinig in waarde stijgen. Het θ Théta stuk is eigenlijk overal goed, maar kan gebruikt worden voor een tactische terugtrekking. Deze moet dus een net wat minder dan gemiddelde stijging hebben in punten. Dat hebben we uiteindelijk zo verwerkt als hieronder staat in de tabellen en de grafiek. Deze wordt berekend voor de stukken van de AI zelf en voor de

● Zéta ● Éta ● Théta

vak	waarde	Stijging
1	100	0%
2	110	10%
3	120	20%
4	130	30%
5	140	40%
6	150	50%
7	160	60%
8	170	70%

vak	waarde	Stijging
1	130	0%
2	131,3	1%
3	132,6	2%
4	133,9	3%
5	135,2	4%
6	136,5	5%
7	137,8	6%
8	139,1	7%

vak	waarde	Stijging
1	200	0%
2	206	3%
3	212	6%
4	218	9%
5	224	12%
6	230	15%
7	236	18%
8	242	21%



tegenstander. Deze worden dan in de uiteindelijke berekeningen van elkaar afgetrokken.

Gemiddelde positie van de stukken

Deze spreekt volgens mij voor zich. De AI pakt de afstand van alle stukken tot het begin van het bord aan zijn kant. Al die waarden worden bij elkaar opgeteld en gedeeld door het totaal aantal waardes om het gemiddelde te krijgen. Dit wordt ook voor de tegenstander gedaan maar dan vanaf de tegenstanders kant van het bord. Dit wordt, net als de waarden van de stukken, van elkaar afgetrokken in de uiteindelijke berekening. De reden dat deze berekening er nog inzit is omdat wij het in Sampi voordelig is om naar voren te bewegen. Het is dus verstandig om ook nog een speciale

berekening te doen voor de afstand en niet alleen te vertrouwen op de toename in waarde van de stukken

Dekking van de stukken

Met weinig stukken in het spel is het belangrijk dat je jouw stukken zoveel mogelijk verdedigt. Daarom hebben we ook nog een berekening erop losgelaten om te controleren hoe verdedigt de stukken zijn. Dit hebben we gedaan door per stuk, elk vlakje te controleren op een ander stuk. Als dit dan een stuk is van de AI zelf, dan gaan we kijken of dat stuk vanuit die positie ook op het vlak van het stuk die we controleren kan komen. Als dat zo is dan is het stuk dus verdedigt. Hier rolt dan een hoeveelheid dekkingen uit. Een stuk kan dus meerdere keren gedekt zijn en dit telt dan ook meerdere keren. Dit is zo omdat hoe meer een stuk gedekt zijn hoe beter het is. Deze berekening doen we, anders dan de andere, niet voor de tegenstander. De reden daarachter is dat de het aantal stukken die de tegenstander gedekt heeft niet veel invloed heeft op wat de beste zet is voor onze AI.

```
// cover thingy
for (int k = 0; k < boardLayout[i].Count; k++) {
    for (int p = 0; p < surroundingTiles.Length; p++) { // looping every surrounding tile
        if (boardLayout[i].Contains(boardLayout[i][k] + surroundingTiles[p])) { // check if piece is surrounded by other piece
            foreach(Vector3 move in piecePrefabs[boardType[i][boardLayout[i].IndexOf(boardLayout[i][k] + surroundingTiles[p])]].GetComponent<Piece>().pieceSO.MoveLocations) { // gather all possible moves for surrounding piece
                if (surroundingTiles[p].x == -move.x && surroundingTiles[p].y == -move.y) { // check if possible move is piece
                    covered++;
                }
            }
        }
    }
}
```

Geen stukken weggeven

De eerste beveiliging, en tevens ook de belangrijkste, is de bescherming tegen het weggeven van gratis stukken. De AI ging er namelijk eerder soms vanuit dat de tegenstander een gratis stuk toch niet zal slaan, en dit is in de meeste gevallen verkeerd. De Manier waarop we dit doen is doormiddel van dezelfde berekening als het uitrekenen van de dekking van een stuk, alleen dan omgedraaid. Hier zoeken we dus juist stukken die niet gedekt zijn. Dit wordt uitgebreid met een soortgelijke berekening maar dan om stukken van de tegenstander te vinden die dit stuk kunnen slaan. Als dit beide het geval is, is het dus gratis weggeven van een stuk en dus een slechte zet. Deze halen we het systeem van de AI door hem niet toe te voegen en deze stap wordt dus uitgevoerd voordat de lijst wordt gevuld met indelingen.

Gratis stukken pakken

Dit is ook een simpele, een gratis is 99% van de keren een goede zet om te spelen. Deze zet hebben we dan ook verplicht gesteld aan de AI. Als de eerste zet er een kan zijn waarbij de AI een gratis stuk slaat, dan zal hij spelen. Dit is gedaan op exact dezelfde manier als de dekking berekening maar dan uitgevoerd op het stuk van de tegenstander.

Zet spelen

Als laatste is natuurlijk zaak dat de AI de zet ook echt speelt. Dit is niet zo makkelijk als gewoon kijken welk stuk op de oude locatie staat en deze dan verplaatsen naar de nieuwe. De AI moet ook nog doorgeven dat het een beurt heeft gespeeld zodat de beurt kan verschuiven naar de tegenstander, net als de beurt indicator. Ook moet er een stuk weg worden gehaald als de AI een stuk van de tegenstander slaat, en dit moet ook verwerkt worden door het hoofd script. Deze zorgt er dan voor dat de variabelen die bijhouden hoeveel er van elk stuk is geslagen en die van de totale stukken geüpdatet worden. Ook moet er als de AI een winnende zet speelt van alles gebeuren. De AI moet doorgeven dat hij heeft gewonnen en dat de winst animaties moeten spelen. In eerdere testen ging dit nog wel een fout leek het alsof wit constant won ook al won zwart meer dan de helft van de potjes.

Hoe leert de AI

Om een AI te zijn moet ons programma officieel ook kunnen leren en aanpassen. Dit hebben wij dan ook toegepast. Dit hebben we gedaan door in het spel een soort analyse functie te verwerken. Deze geeft na een potje aan de AI door wat hij goed en of fout heeft gedaan. Neem bijvoorbeeld een AI die een potje lang alleen maar stukken naar voren speelt en onverdedigd laat. Als deze dan verliest geeft de analyse door aan de AI dat dingen onverdedigd laten onverstandig is, en dat ie misschien minder snel naar voren moet. Dan past de AI de waardes van hoe belangrijk sommige delen van de berekening zijn aan naar wat hij te horen krijgt van het spel. Hij past zich hierdoor aan en ontwikkelt steeds een betere speelstijl. Deze functie hebben we er echter uitgehaald in de publiek beschikbare versie nadat de AI een redelijk niveau had bereikt. De reden hierachter is dat we de spelers een constante ervaring wouden geven zodat ze het spel ook vaak zouden kunnen spelen zonder dat de AI zich aanpast aan de Speler en uiteindelijk onverslaanbaar wordt. Of juist met een verandering van techniek heel slecht wordt. We trainen de AI zelf en update dan het spel met de nieuwe AI.
Hieronder zijn de waarde die de AI nu heeft zichtbaar. En de formule die de AI hanteert bij het berekenen van hoe goed een zet is.

$$TotalBoardValue = ((totalPieceValue * W\_pieceValue)\,(diffTotalPieceValue * W\_diffPieceValue))\,W\_Values) + (((averageBoardPosition * W\_averagePosition)\,(diffAverageBoardPosition\,W\_diffAveragePosition)) * W\_Positions) + (covered * W\_coverage));$$

W_piece Value	1.017
W_diff Piece Value	0.983
W_average Position	0.862
W_diff Average Position	0.906
W_coverage	1.095
W_Values	1.628
W_Positions	0.981

Ontwerpcyclussen

1

Op papier

We zijn begonnen in de trein naar delft met pen en papier. We besloten al snel dat we iets wilden met veel tactiek en zonder een gelukfactor. Daarna hebben we nagedacht over hoe het bord er uit moest komen te zien. We gingen voor een spel waar je naar de overkant moet lopen. Het bord moest niet te groot worden, want dan duurt het te lang om naar de overkant te komen. Het bord moest ook niet te klein zijn dan ben je namelijk binnen een paar zetten aan de overkant. Vervolgens moesten we een beslissing nemen over de breedte van het bord. We wilden het bord zo smal mogelijk maken zodat je niet makkelijk om elkaar heen kan. Je moet echter wel om elkaar heen kunnen. Het is namelijk wel de bedoeling dat je langs elkaar kan lopen zonder dat je elkaar hoeft te vermoorden.

Daarna hadden we nog stukken nodig om mee te bewegen. We besloten om verschillende stukken te maken die naar voren, naar achteren en naar de zijkant kunnen lopen. Het is dan namelijk mogelijk dat bepaalde stukken van meerdere kanten kunnen aanvallen.

prototype

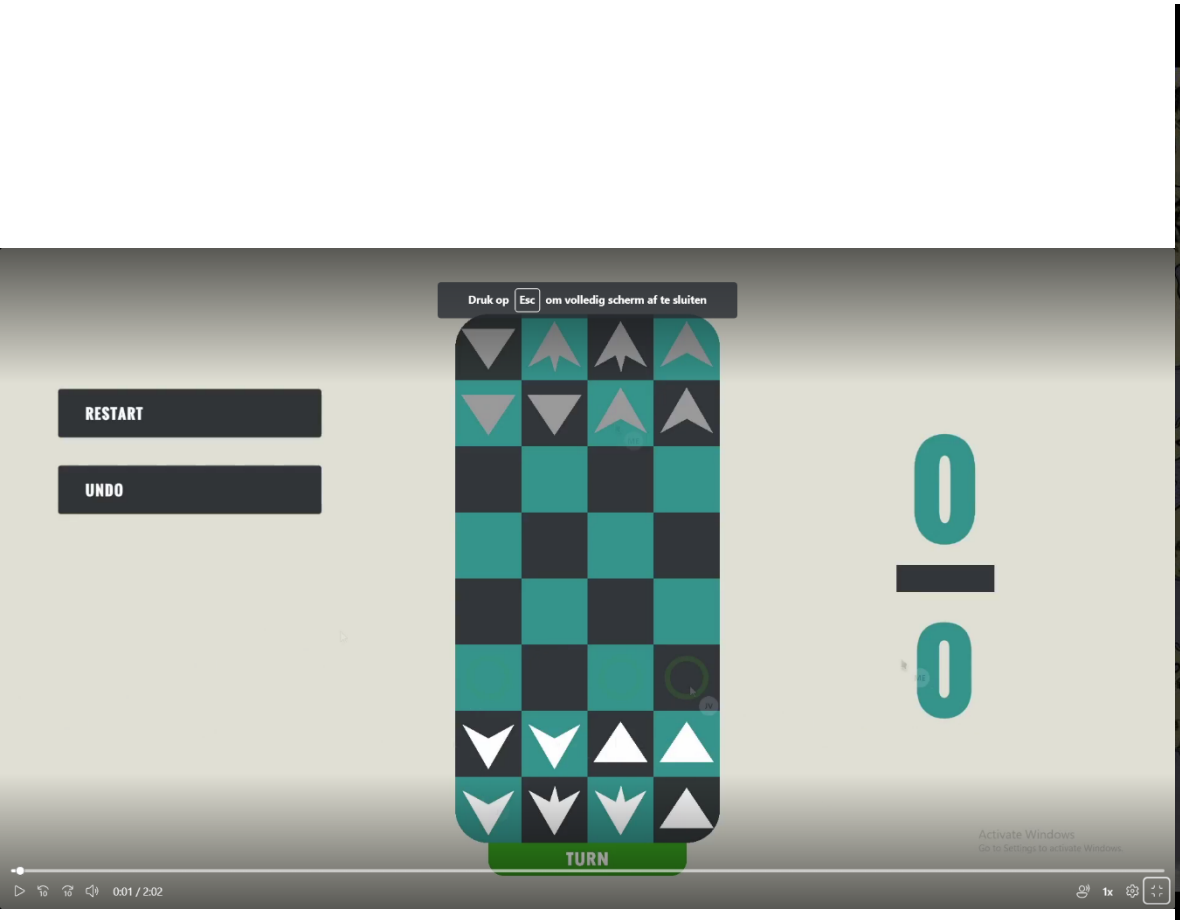
Nadat we bovenstaand idee verder hebben uitgewerkt hebben we een prototype gemaakt op de app. Het logo van de app staat hier rechts. We zijn toen begonnen met het maken van het bord. We hebben gekozen voor 8 x 4 vlakken. Daarna hebben we gekozen voor een stuk die alleen maar naar voren kan lopen. Deze hebben wij het symbool van een speer gegeven. Dit is ook te zien op de bladzijde hieronder. We wilden ook een stuk wat ook opzij zou kunnen zodat dit stuk ook kan verdedigen. Daarnaast vonden we dat we nog een stuk nodig hadden. Dit stuk moest het sterkste worden. We hebben hier langere tijd over nagedacht. Omdat dit stuk anders moest worden dan de andere stukken. We hebben geprobeerd wat we er van vonden als dit stuk ook achteruit kon lopen. Vervolgens hebben we alles in de app gedaan. Toen zijn we gaan spelen.

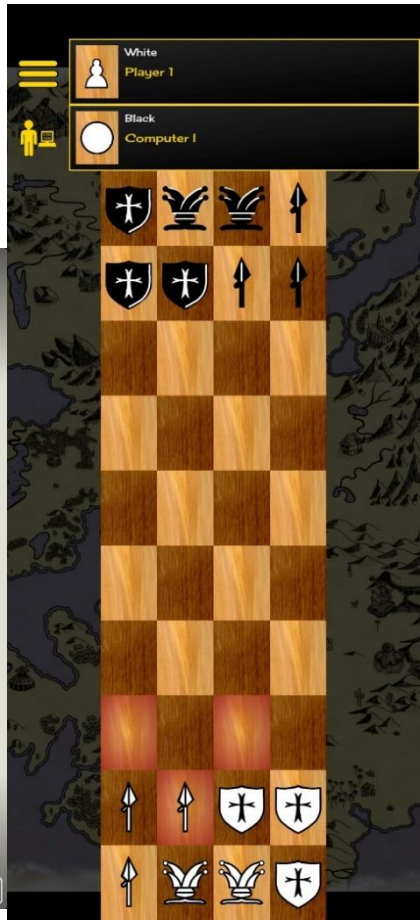
Op de computer

Toen we hadden besloten hiermee verder te gaan zijn we van het prototype naar een eigen spel gegaan. We hebben dit gedaan in Unity met de codeer taal C#. We hebben het hier ingedaan, omdat wij hier dan ook de AI in konden maken. Als eerste hebben we het bord ontworpen. Voor de kleuren hebben we voor vlakken in zeegroen en donkerblauw-grijs gekozen. Aan de punten van het bord hebben we een kwart cirkel toegevoegd, hierdoor is het bord minder puntig op de hoeken. Hiermee was het ontwerp van het bord klaar. Vervolgens hebben we nagedacht over de stukken. Ze moesten duidelijk van elkaar te onderscheiden zijn. We hebben gekozen voor stukken met een punt die wijst in de richting van de vlakken waar het stuk naar toe kan en mag bewegen. De kleur van de stukken moesten een contrast hebben met de kleuren van het bord. We hebben gekozen voor witte en grijze stukken.

AI

voor het maken van de AI zijn we bezig geweest met het zorgen dat de AI werkt. Hij moest de stukken kunnen bewegen en hij moest weten waar ze heen gaan. Hierna hebben we getest of hij dat kan hij kan het maak hij moest toen gaan uitreken hoeveel mogelijke posities er waren. Toen hebben we een algoritme geschreven die kon bepalen hoe goed een stelling is. toe was het aan de AI om te leren en de waarden van ons algoritme perfect te maken.





2

Bordspel

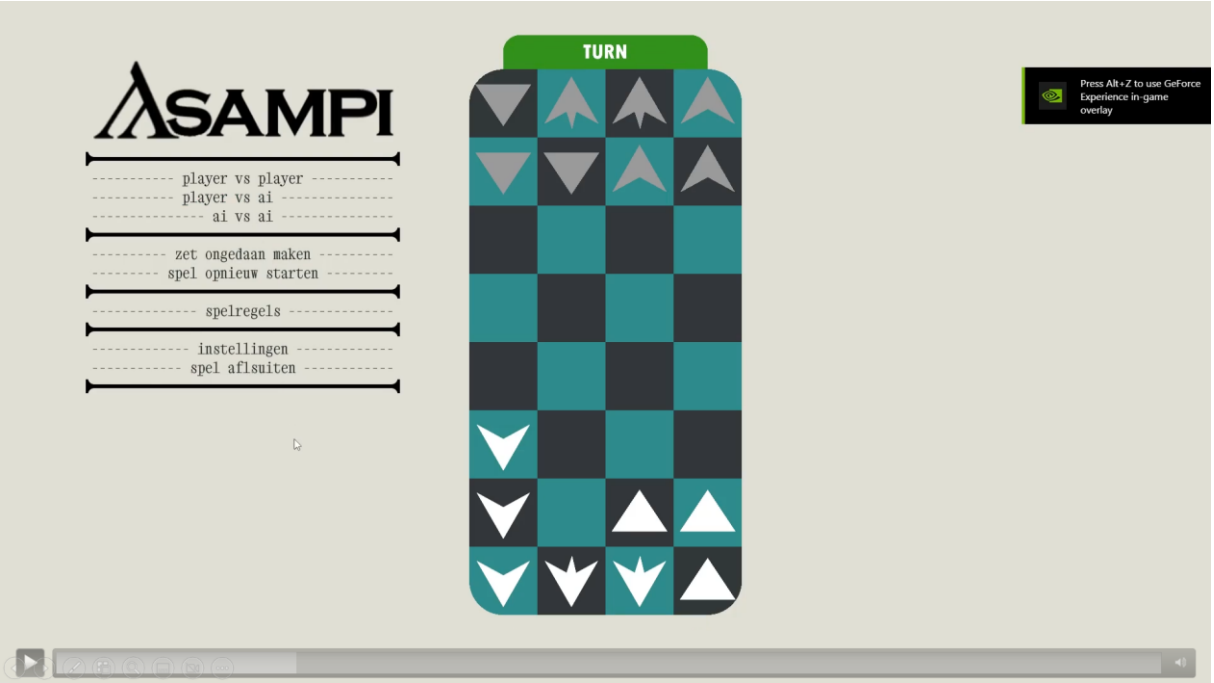
Uit de eerste testen van het spelen van het spel kwam naar voren dat de stukken goed te onderscheiden waren en de kleuren ook. De spelregels moesten nog wel aangepast worden. Eerst hadden we een punt voor elk stuk wat er aan de overkant kwam en voor alle stukken die nog op het bord stonden. Dit werkte niet omdat hierdoor niemand naar de overkant ging. Je kon namelijk beter je stukken bij je houden en met meer stukken het spel eindigen. Naar aanleiding hiervan hebben we de spelregels veranderd. Nu krijg je voor ieder stuk die de overkant bereikt en punt en het stuk zou dan de andere kant op lopen zodat hij de tegenstander in de rug zou aanvallen.

Computerspel

Het eerste computerspel wat we hadden was eigenlijk alleen het spel. We moest dit veranderen. We moesten er voor zorgen dat je meer keuzes zou hebben hoe je speelt. Daarom hebben we de player VS player, AI VS player en AI VS AI. toegevoegd zodat je kon wisselen tussen de verschillende soorten in het computerspel. Ook moesten we de spelregels toevoegen. Anders wisten mensen niet hoe ze het spel konden spelen. Ook hebben we een knop toegevoegd om het spel te stoppen als je niet meer verder wil spelen. Eerst moesten we namelijk steeds op esc klikken.

AI

Bij het testen van de AI kwamen we er al snel achter dat de AI stukken weg gaf. Dit is natuurlijk niet handig en wij vroegen ons af waarom dit was. We kwamen erachter dat de AI van de tegenstander een domme zet verwachte en dat hij daarop inspeelde omdat hij hierdoor beter uitkwam. Dit hebben we aangepast door een soort beveiligingen in te bouwen zodat de AI een domme zetten zou maken. We hebben er namelijk voor gezorgd dat de AI niet meer stukken weggeeft. En als er een stuk gratis gepakt kan worden, pakt de AI die verplicht.



3

Bordspel

Ook de spelregels van de tweede ontwerpcyclus waren niet echt goed, bij het testen zagen we dat mensen niet naar de overkant gingen omdat het spel al klaar was voordat je aan de overkant was gekomen. Als je dan aan de overkant kwam was het weer zodat de tegenstander ook aan de overkant kwam waardoor je eigenlijk geen verschil had met als hij de afging. Hierdoor hebben we besloten dat als je aan de overkant komt dat je dan gelijk hebt gewonnen. De testen die we met deze regels hebben gedaan gingen goed. Op de PWS-middag hebben we dit ook nog getest en iedereen was er erg tevreden mee. De spelregels en de stukken van ons spel zijn door de mensen van de PWS-middag goedgekeurd.

Computerspel

Voor het computerspel hebben we dus de spelregels aangepast zodat het spel leuk en goed te spelen is. Ook vonden wij het belangrijk dat je kan zien waar de stukken naar toe kunnen. Dit hebben we gedaan zodat het spel wat duidelijker wordt voor de spelers. Daarnaast vonden we het handig dat je kan zien hoeveel stukken er van het bord af zijn, dit is moeilijker te zien als de stukken verder uit elkaar staan.

SPELREGELS

1 De stukken mogen alleen in hun richting lopen en niet van het bord af lopen. Je kunt de richting van waar de stukken heen kunnen zien in het begin scherm onder de stand en wie welke kleur is.

2 Jouw stukken kunnen niet op ander stuk van jouw landen.

3 Als jouw stuk op het stuk van de tegenstander komt sla je deze en wordt die uit het spel verwijderd.

4 Als een van de stukken naar de andere kant van het bord is gelopen kan deze op de volgende zet van het bord af lopen en heeft de speler van wie het stuk is gewonnen

5 Als je alle stukken van de tegenstander geslagen hebt betekent dit dat je automatisch hebt gewonnen omdat de tegenstander geen zetten meer kan doen

6 Als jij jouw stuk van het bord af speel heb je gewonnen ook al heb je geen stukken meer om mee te spelen

SAMPI

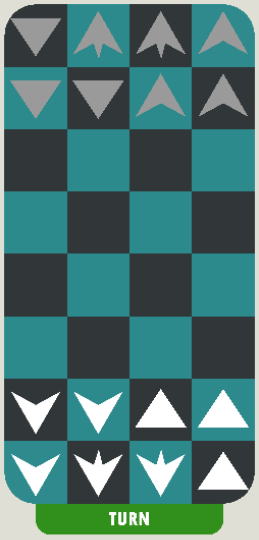
---------- player vs player ----------
---------- player vs ai ----------
---------- ai vs ai ----------

--------- spel opnieuw starten ---------

spelregels

---------- spel afsluiten ----------

Programming: Jonas Valstar
Design: Jonas Valstar, Matthijs Eikelaar
Rules: Jonas Valstar, Matthijs Eikelaar
versie: prototype-3



WIT	ZWART
GESLAGEN STUKKEN	GESLAGEN STUKKEN
x 0	x 0
x 0	x 0
x 0	x 0

LEGALE ZETTEN