- Beter: elke cel heeft 2 muren, boven & links. Voor rechter/ondermuur kijkt hij naar cel onder zich of cel naast zich. Als het een randcel is is de rechter/ondermuur per definitie gesloten.

- Muurgroeier (weet niet of deze al bedacht is?) - CPU-intensief + Zelf bedacht (voor zover ik kan overzien/herkennen)
    - Buitenkant maze is een en al muur
    - Zet muur op willekeurige plek neer, aan de zijkanten (muren) vast. Herhaal:
        - Volgende muur moet aan de muren die er al zijn vastzitten, maar mag niet aan beide kanten aan een muur vast zitten
        - Als er geen plekken meer zijn die aan een kant een muur hebben zitten en aan een andere wel is de maze af.
        - Je kan elke muur een eigen variabele geven, die aangeeft of hij grenst aan muren of niet. Als je dan een muur plaatst, dan herevalueer je voor alle aangrenzende muren of ze nu aan 2 muren grenzen. Als dat zo is pas je die variabele aan, anders niet.
    - Profit!
- Hunt & Kill algoritme (Wat anders moet: i.p.v. dat ie backtracked, kiest ie gewoon een willekeurige cell die nog niet bezocht is.) + Weinig geheugen - Doolhoven zijn niet heel erg bijzonder maar voor simpele toepassingen goed genoeg
    - Neem punt als startpunt. Markeer deze als bezocht. Alle andere cellen zijn nog niet bezocht.
    - Schakel naar plot-modus. Herhaal:
        - Plot-modus
            - Zijn er onbezochte hokjes aangrenzend aan het huidige hokje?
                - Maak een willekeurig hokje het huidige hokje en maak een verbinding tussen het vorige hokje en het huidige hokje. Markeer het huidige hokje als bezocht en voeg het toe aan De Lijst.
                - Anders: schakel naar zoek modus
        - Zoek-modus
            - Verwijder het huidige hokje uit de lijst.
            - Maak het vorige hokje het huidge hokje.
            - Zijn er onbezochte hokjes aangrenzend aan het huidige hokje?
                - Schakel naar plot-modus
                - Anders: is het huidige hokje het starthokje?
                    - Einde algoritme
- Binair doolhof + Snelst/minste geheugen - lelijk & makkelijk
    - Voor elke cel
        - Zorg dat hij naar rechts of naar beneden leidt. Dat betekent dat van elke cel dus óf de rechtermuur (leidt naar beneden) óf de ondermuur (leidt naar rechts) getekend mag worden.
        - De meest bovenste mag óf onderste muur óf geen muur (en dan de bovenste muur wel anders is de doolhof open)

        - De meest linker muur mag óf rechtermuur óf geen muur (en dan de linker muur wel want anders is de maze open
- Prim's algoritme + Startpunt vinden is makkelijk, maar twee willeukeurige punten is moeilijker - Ze hebben een duidelijke richting (alle gangen gaan een beetje in een bepaalde richting)
    - Kies een cel als startpunt.
    - Maak alle cellen niet onderdeel van het doolhof.
    - Voeg de cellen aangrenzend aan het startpunt toe aan de front-lijst.
    - Herhaal totdat alle cellen deel zijn van het doolhof:
        - Kies een willekeurig front
        - Kies een van de aangrenzende cellen die nog geen onderdeel zijn van het doolhof.
        - Als deze ook nog aangrenzende cellen heeft die nog geen onderdeel zijn van het doolhof maak het dan ook een front.
        - (Kies weer de willekeurig gekozen cel) als deze nog aangrenzende cellen heeft die nog niet bij de maze hoort blijft het een front, anders mag ie uit de front-lijst.

- Dijkstra's:
    - Gedachte: ~~Veld met cellen. Elke cel kan een paar states zijn: front, bezocht en niet bezocht. Markeer het startvakje als front. Dan itereer je steeds over alle cellen heen: bij een front maak je alle onbezochte buurcellen fronts, laat je de huidige cel parent van de front-gemaakte cellen (zo kun je de weg terugvinden) en je geeft de front gemaakte cellen een afstand van de huidige cel tot het begin + 1. en maak je het zelf een bezochte cel. DOorgaan totdat elke cel gehad heeft.~~ Hetzelfde, maar dan met lists. Elke keer alle cellen checken is TE VEEL GEKTE. Pros: Checkt alle paden, OOK DE GEKKE Cons: vunzig traag? Omdat ie écht alle paden checked.
    - Niet vergeten om:
        - Te checken voor omliggende cellen die een kortere afstand hebben tot het startpunt
        - Als er een route is gevonden, KLAAR want bij (perfecte) doolhoven is maar 1 route mogelijk!
        - => Het algoritme dus goed te begrijpen en goed te implementeren

Indeling van het programma:

- Intro? :D =>
- Hoofdmenu:
    - Genereer doolhof
    - Uitleg algoritmen
    - Over
    - Afsluiten
- Genereer doolhof.
    - Kies algoritme:

            - Muurgroeier
            - Hunt & Kill
            - Binair
            - Prim's
        - Grootte
            - Klein [5/5]
            - Gemiddeld [20/20]
            - Groot [50/50]
        - Snelheid
            - Langzaam [10 iteratie p/s]
            - Gemiddeld [100 itertaies p/s]
            - Snel [200 iteraties p/s]
        - Genereer!
- Doolhof scherm:
    - Toggle oplossing aan/uit
    - Snelheid toggle Pauze/Langzaam/Gemiddeld/Snel
    - Terug naar hoofdmenu knop
    - Genereer nog een doolhof van dit type
    - Het gegenereerde doolhof
- Uitleg algoritmen (Elk met een animatie, generatie algoritme klein & langzaam?):
    - Muurgroeier
    - Hunt & Kill
    - Binair
    - Prim's
    - Dijkstra's
- Over
    - Verhaaltje over wat het plan was, hoe ik het gedaan heb en wat ik van het resultaat vind.

Maze gamestate:

- De maze gamestate
    - Logic
        - maze.logic(int tempo, bool solution); geeft SDL_surface* met maze + evt. oplossing erop. Regeld zelf timing etc.
            - Bestaan 4 types voor: (inheritance dus)
                - Prims
                - Muurgroeier
                - Binair
                - Hunt & Kill
            - Elk heeft van zichzelf standaard een maze met in het begin alle muren gesloten, alle velden niet bezocht (overal zwarte streepjes, grijs). Zodra Cellen bezocht zijn worden ze wit. Muurjtes worden weggehaald volgens het algoritme. Ook nog iets implementeren met dijkstra's (dat cellen die ie bezocht heeft rood kleuren ofzo)

            - Standaard functies: logic & render
            - Het goede pad wordt
        - Bewerkt het surface dat het er goed uitziet. Wanneer te klein vergroten e.d. Bij max grootte gewoon pasten.
        - Handles de UI (sneller/langzamer/pauze, oplossing laten zien ja/nee, terug naar het menu
    - Render

Bron:
http://en.wikipedia.org/wiki/Dijkstra's_algorithm
http://en.wikipedia.org/wiki/Maze_generation_algorithm
http://www.astrolog.org/labyrnth/algrithm.htm#perfect
http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap

Niet implementeren:

- Kruskals algoritme
    - Twee lijsten: bezocht en niet bezocht, en in de bezochte lijst hou je ook bij tot welke set ze behoren..
    - Itereer totdat alle cellen tot een set behoren
        - Kies steeds horizontaal of verticaal twee aangrenzende cellen. Dit kan door uit de lijst niet bezocht een willekeurige cel te kiezen. Kies van deze cel een aangrenzende cel. Behoort deze cel tot een set?
            - Verbind de huidige cel met die cel en voeg hem aan de set toe. Veplaats de huidige cel naar de bezochte lijst.
            - Anders: Maak een nieuwe set met de andere cel. Veplaats de cellen naar de bezochte set.
    - Op een gegeven moment hebben alle cellen een set. Maar ze zijn nog niet dezelfde set. Nu moet je alle cellen dezelfde set maken.
    - Herhaal totdat alle cellen dezelfde set hebben.
        - Maak een lijst met alle cellen die aan meerde sets grenzen.
        - Kies uit die lijst een willekeurige cel, en verbind die cel met een willekeurig aangrenzende set.
        - Voeg de aangrenzende set aan de set van de huidige cel toe.
    - Klaar!
- A* algoritme
    - Toch maar niet? Aangezien een pad niet rationeel is: het pad zal vaker terugaan en dan vooruitgaan dan dat het een logische richting is. Dijkstra's is misschien beter: dat vind een pad en let niet op hoe logisch het is of niet. Prestatie is niet echt een issue hier, aangezien beide algoritme er toch wel lang over doen omdat er altijd vage paden zullen zijn. Als het allemaal soepel gaat kan ik alsnog het algoritme implementeren, maar daar reken ik nu niet op.