ChatGPT

Digital Blood on the Clocktower Game Engine & Script-Difficulty Scoring

Overview of the Table-top Game

Blood on the Clocktower is a social-deduction game where five to twenty players sit in a circle and take on hidden roles. Each player draws a character token that tells them whether they are **good** (blue) or **evil** (red) and explains their special power [1]. Play alternates between **day** (open eyes) and **night** (eyes closed) phases [2]:

- **Day:** Players talk publicly or privately, share information, bluff and lie, and nominate someone for execution. If a nominated player receives at least half of the living players' votes, they are executed [3]. Dead players may still talk but may only vote once more in the entire game [4].
- **Night:** All players close their eyes. The Storyteller (moderator) wakes specific characters in a fixed order so they can use their ability or receive information [5]. Signals (taps, finger gestures, etc.) are used to communicate silently during the night [5].

A game ends when either the **Demon** (evil) dies—giving the good team victory—or when only two players remain alive, in which case the evil team wins [6].

Characters and Alignments

Characters are categorised into several broad types:

Type	Description	Examples
Townsfolk (good)	Provide useful information, protection or actions that help the good team. They lose their ability when dead, drunk or poisoned [7].	Washerwoman, Empath, Monk, Slayer
Outsiders (good but hindrance)	Have abilities that usually harm their team or restrict their actions (e.g. Butler may only vote when a chosen player votes; Drunk thinks they are a Townsfolk but actually have no ability).	Butler, Drunk, Saint [8]
Minions (evil)	Support the Demon by poisoning, providing misinformation or other disruptive abilities.	Poisoner, Spy, Scarlet Woman [9]
Demons (evil)	The main villain; kills a player at night and often has an additional power. If the Demon dies, good wins [6].	Imp (basic kill once per night), Vortox (kills and makes all information false) [10], Po (kills three players on alternate nights)

Type	Description	Examples
Travellers	Extremely powerful characters used for players who arrive late or must leave early. They have strong abilities but gain little information [11].	
Fabled	Characters used by the Storyteller to adjust real-world issues (e.g. speeding up a game, assisting players with disabilities). Fabled are not intended to add spice but to ensure a smooth game [12].	Angel, Sentinel, Deus ex Fiasco [13].

Each player maintains independent **states** that can change over time: alive/dead, good/evil, sober/drunk, healthy/poisoned, and mad/not mad [14]. These states are independent: a player may be drunk and poisoned simultaneously [15]. *Drunk* or *poisoned* players lose their ability; they still wake and act but the ability has no effect, and information given may be false [16]. Alignment and character can change independently; for example, a good player may have an evil character or vice versa [17].

Editions and Difficulty Levels

The base game contains three editions, each designed to provide a different play experience. The official descriptions of these editions and their suggested skill levels will guide the difficulty scoring algorithm later.

Edition	Summary & difficulty	Evidence
Trouble Brewing (beginner)	A straightforward Demon-hunt where good characters gain information and evil tries to muddy the waters. The rules describe it as "relatively straightforward" and state that *Trouble Brewing* is *recommended for players and Storytellers new to Blood on the Clocktower or to social deception games* [18].	The edition page states that it provides a variety of information sources and is labelled *Beginner* [18].
Sects & Violets (intermediate/ crazy)	The wildest edition—players receive huge amounts of information while demons and minions create chaos. Characters can change alignment or even swap characters. The page notes that this edition is "the craziest of the three", that information may be false depending on the Demon, and it introduces madness [19]. It is *recommended for players who want to do wild and unexpected things* [19].	Official description emphasises that good players must sort true from false information and that there are alignment and character swaps [20].

Edition	Summary & difficulty	Evidence
Bad Moon Rising (intermediate/ violent)	A death-heavy edition where Demons kill multiple times and Minions can kill too, but there are many protective and resurrection abilities. It is labelled *Intermediate* and recommended for proactive players who don't fear dying [21] .	The edition page calls it a "death extravaganza" with high kill rates and notes that players need to figure out which Minions and Demons are in play to survive [22] .

Special modes like **Teensyville** cater to five or six players, featuring a small script of six Townsfolk, two Outsiders, two Minions and two Demons [23] . These smaller games make it easier to track which characters are in play but still allow combinations from any edition [23] .

Key Mechanics for a Digital Engine

Setup and Role Distribution

Setting up a game requires choosing an edition, selecting the appropriate number of Townsfolk, Outsiders, Minions and Demons based on player count, and secretly assigning these to players. For example, an eight-player **Trouble Brewing** setup might include 5 Townsfolk, 1 Outsider, 1 Minion and 1 Demon [24] . Some characters modify the composition: the Baron adds two Outsiders and removes two Townsfolk [25] , and the Drunk or other tokens can cause similar changes [26] . Fabled and Travellers are generally not used in a first game but may be added when needed [27] .

Day and Night Phases

The digital engine must alternate between **day** and **night** phases. During the day, players discuss, bluff and nominate. Executions happen after a vote requiring at least half of the living players' votes [3] . The engine should implement a voting system that records nominations, tracks votes (dead players only have one vote remaining), resolves ties (no execution on a tie) and applies the consequences.

During the night, the engine wakes characters in the **night order** to perform their ability. The official night sheet lists the order for every script; although many characters wake, abilities may trigger immediately and sometimes outside this order [28] . Implementing night order in code involves assigning each character an integer **nightOrder** property and iterating through sorted characters each night. Some abilities (e.g., the Ravenkeeper using their ability immediately upon death [29]) need event-based triggers; the engine therefore requires a flexible event system.

States and Ability Resolution

Each player's **state**—alive/dead, alignment, drunk/poisoned, mad/not mad—must be tracked independently [14] . The engine should support the following rules:

- **Ability loss:** Players lose their ability when dead, drunk or poisoned; persistent effects end immediately [30] . When sober and healthy again, they regain their ability, but one-off abilities used while drunk remain expended [31] .

- **Information secrecy:** Only tell a player what their ability specifies; other players should not learn what happens unless the rules say so [32] .
- **Alignment and character changes:** Alignment and character are independent. When a player's character or alignment changes, they learn it at the earliest opportunity; this information is not subject to drunkenness or poisoning [33] .
- **Drunkenness and poisoning:** Drunk or poisoned players have no ability but think they do; the engine may give them false information [16] . Drunk or poisoned states do not cancel each other [15] .
- **Madness:** Players may be instructed to act "mad" about something. If they do not behave accordingly, the Storyteller can penalise them (e.g., execution). Madness is not a visible game state but must be enforced by judging players' messages [34] .

Generic Patterns of Character Abilities

Across editions, abilities can be grouped into patterns that the engine can implement generically. Each ability is represented as a function or set of rules that runs in response to triggers (night start, night end, during the day, on death, etc.). Patterns include:

1. **Information:** Character gains information about other players' alignment or character (e.g., Washerwoman learns that two players include a Townsfolk). Implementation: engine shows the player specific character tokens, finger numbers or yes/no signals.
2. **Protection/Prevention:** Prevents a chosen player from dying or being targeted (e.g., Monk protects a player; Tea Lady protects neighbours). Implementation: engine marks protected players; kills targeting them fail.
3. **Poison/Drunk:** Causes a player's ability to be lost or information to be false (e.g., Poisoner, No Dashii). Implementation: engine flags targets as poisoned for a duration.
4. **Kill:** Character kills one or more players (e.g., Imp kills one; Po kills three on alternate nights). Implementation: engine marks players for death at dawn. Some demons also have "swap" features (Imp can pass the demon token to a Minion).
5. **Alignment/Character Change:** Changes a player's alignment or character (e.g., Pit-Hag turns players into new characters; Snake Charmer swaps with the Demon). Implementation: engine replaces character object for player and updates alignment accordingly.
6. **Resurrection/Revival:** Brings a dead player back (Professor). Implementation: engine resurrects target and restores ability.
7. **Vote/Execution Effects:** Special behaviour when nominated or executed (e.g., Saint's ability kills the whole good team if executed, Virgin's ability kills the first player who nominates them). Implementation: engine intercepts nomination/execution events and applies effect.
8. **Information Manipulation:** Forces information to be false (Vortox), hides information (Spy learns the entire Grimoire), or globally changes how information works. Implementation: engine uses a flag in the global state (e.g., `infoIsFalse=true`) when Vortox is alive [10] .
9. **Madness Enforcement:** Characters like Cerenovus instruct players to be mad about certain topics; the engine must track whether players follow the instruction and apply penalties if not [34] .

These patterns allow developers to write generic modules (e.g., `PoisonEffect`, `KillAction`) and attach them to characters, rather than hard-coding each ability.

Proposed Digital Architecture

High-Level Components

1. **Client Application** – provides user interfaces for players (web or mobile). It displays character sheets, allows chat and private whispers, handles nominations and votes, and presents night prompts to players when they must act. A spectator/staff UI is used by the Storyteller.
2. **Game Server / Engine** – authoritative process that manages game state and enforces rules. It keeps track of players, characters, alignment, states (alive, drunk, poisoned), the current phase, night order, and resolves abilities. It sends prompts to clients, collects decisions, and applies effects.
3. **Data Store** – persists games, scripts and character metadata. Character definitions include ability text, tags (information, kill, misinform, protect, align-change, etc.), weight for difficulty scoring, allowed phases and night order. Scripts are lists of character names plus configuration (player count, edition, optional travellers/fabled).
4. **Script Manager & Difficulty Calculator** – creates, validates and scores scripts. It ensures the correct number of characters per alignment and applies modifications from characters like Baron. It calculates a difficulty rating using the algorithm described below.
5. **Communication Layer** – handles messaging between players, including public chat, private whispers, and Storyteller prompts. Enforces “no peeking” by hiding restricted information.
6. **AI/Storyteller Assistance Module** – optional logic to assist or automate Storyteller decisions for characters that require a choice by the Storyteller (e.g., which player a kill targets if the Demon cannot decide). It can implement deterministic selection (e.g., kill the player with the lowest ID) or use heuristics to ensure reproducibility.

Data Model Outline

Below is an outline of key classes or structures the server should maintain. For clarity, pseudocode uses a modern typed language (TypeScript/Java); actual implementation may vary.

```
class Player {
    id: string
    seat: int
    userId: string
    character: Character
    alignment: Alignment
    alive: boolean
    drunk: boolean
    poisoned: boolean
    madAbout: string | null
    voteTokens: int  // 1 for dead players, unlimited for alive players
}

class Character {
    name: string
    type: CharacterType  // Townsfolk, Outsider, Minion, Demon, Traveller,
Fabled
    edition: Edition
    tags: Set<Tag>
    nightOrder: int[]  // positions in first night and other nights
    ability: AbilityFunction
    weight: DifficultyWeight
}

class GameState {
    players: List<Player>
    phase: 'day' | 'night'
    dayCount: int
    nightCount: int
    script: Script
    infoIsFalse: boolean  // global flag when Vortox alive
    events: EventQueue
    // Additional flags: e.g., demonStarPassUsed, etc.
}

class Script {
    name: string
    edition: Edition
    characterNames: List<string>
    travellers: List<string>
    fabled: List<string>
}
```

Event System & Deterministic Resolution

The **EventQueue** drives the game. Each night the engine builds a list of events in ascending order of `nightOrder`. For each character still alive and not drunk/poisoned (unless the ability triggers when dead or drunk), the engine creates an event representing their ability. When executed, an event may schedule additional events. Key features include:

- **Deterministic ordering:** Characters act in a known order; if multiple characters share the same order number, tie-break by seat number or predetermined priority. Events triggered by ability effects (e.g., Ravenkeeper after death) execute immediately or at specified offsets [28].
- **Prompt collection:** When an ability requires a choice, the event sends a prompt to the appropriate player (or Storyteller) and waits for a response. A timeout and default decision rule may be used to ensure determinism.
- **State mutation:** When an event executes, it mutates the `GameState`: kills players, assigns poison, changes alignment, etc., and records logs for auditing.
- **Interleaving day events:** After the night queue is empty, the engine increments the day, announces deaths to all players, and enables nominations/voting. When the day ends (e.g., after a successful execution or if players decide not to execute), the engine transitions to night again.

Handling Player Communication and Confidentiality

To replicate the tabletop experience, the digital game must enforce confidentiality. The engine should:

- Prompt only the acting player at night; others see a sleeping screen. Drunk or poisoned players should still receive prompts, but the results may be false [16].
- Prevent players from viewing the Grimoire or other players' tokens [35].
- Support public chat and private whispers during the day; maintain logs for reconstructing decisions if disputes arise.
- Handle madness (behavioural enforcement) by tracking whether players follow instructions and allow the Storyteller to impose penalties [34].

Script Difficulty Scoring Algorithm

Designing a custom script is challenging because the combination of characters influences how hard the game is to play and run. The official editions provide a baseline: *Trouble Brewing* is beginner friendly [18], while *Sects & Violets* and *Bad Moon Rising* are intermediate [19] [21]. A digital system should estimate the difficulty of any custom script using a deterministic algorithm so that Storytellers and designers can balance their games.

Character Tagging & Base Weights

Each character is tagged with qualitative categories that reflect how it influences complexity. The table below proposes tags and base weights for use in the algorithm. A positive weight makes a script harder; a negative weight makes it easier (information-rich scripts are generally easier). The weights are heuristics derived from official edition descriptions (e.g., Vortox making all information false [10], multiple kills in Bad Moon Rising [22], etc.) and community script-building advice [36].

Tag	Description	Weight
info	Provides reliable information to the good team (e.g. Empath, Fortune Teller). Information helps the good team and makes the script easier.	−1
onceInfo	Provides a one-off piece of strong information (e.g. Slayer, Saint proves identity).	−0.5
misinfo	Generates false or ambiguous information (e.g. Poisoner, Vortox).	+2
kill	Kills a player at night or during the day (per kill beyond baseline). Each additional kill increases pressure on the good team.	+1 per extra kill above one per night
multiKill	Kills more than one player per activation (Shabaloth, Po). Added to the base kill weight.	+2

Tag	Description	Weight
protect	Prevents death or revives players (e.g. Monk, Tea Lady, Fool). Protection generally slows the game and gives the good team resilience.	−1
revive	Brings back a dead player (Professor). Reduces pressure on good team.	−2
alignChange	Changes a player's alignment (e.g. Fang Gu, Goon). Very confusing.	+3
charChange	Changes a player's character (Pit-Hag, Snake Charmer). Requires the engine and players to track new abilities.	+2
madness	Introduces madness; players may be penalised for how they talk (e.g. Cerenovus). Adds social complexity.	+2
traveller	Travellers can join/leave at any time and have powerful abilities [11]. They add unpredictability.	+1 per traveller
fabledHelp	Fabled characters that make the game easier (e.g. Angel protects a player).	−1
fabledChaos	Fabled characters that deliberately add chaos or allow mistakes (e.g. Doomsayer, Deus ex Fiasco [13]).	+1

Characters can have multiple tags. For example, the **Vortox** (Demon) would have `kill` (+1), `misinfo` (+2) and a global flag causing all information to be false [10], so its base weight might be around **+4**. The **Empath** would have `info` (−1). The **Pit-Hag** (Minion) can change characters each night, so `charChange` (+2) and implicitly `misinfo` because it can turn good players into something else.

Synergy Adjustments

Certain combinations of tags amplify difficulty:

1. **Multiple misinformers:** If more than one misinfo tag is present in the script (e.g. Poisoner + Vortox), add +1 for each extra misinfo character because overlapping false information makes solving the game much harder.
2. **Multiple kill sources:** For each additional kill tag beyond the Demon (including Minions with kill abilities), add +0.5 because multiple kills reduce time for good players to analyse information.
3. **Alignment and character change interplay:** If both `alignChange` and `charChange` are present, add +1 because players must track both evolving alignments and characters.
4. **Protection vs kill balance:** If the total protection weight (sum of `protect` and `revive` values) exceeds total kill weight, subtract −1 from the final score (game leans easier); if kill weight significantly exceeds protection (kill − protect > 3), add +1.
5. **Information density:** Compute a ratio of (number of info tags) to (misinfo tags). If there are at least twice as many `info` as `misinfo`, subtract −1 (makes the game easier). If misinfo is equal or greater, add +1.

Difficulty Score Computation

For a script **S** with characters $C_1, C_2, \ldots, C_n$:

1. **Base Score:** Sum the weights of all tags on all characters.
2. **Synergy Penalties/Bonuses:** Compute adjustments using the synergy rules above.
3. **Normalisation:** Divide by the number of players (to account for larger games naturally being more complex) and multiply by a constant (e.g. 5) to scale to a convenient range.

4. **Categorisation:** Map the final score to a difficulty category:

5. **Beginner ($\leq$ 5):** The script has mostly information-providing characters, few misinformers and minimal complexity. Comparable to *Trouble Brewing* [18] .

6. **Intermediate (5–10):** Balanced information and mischief; includes some multi-kill or alignment-change effects. Comparable to *Sects & Violets* and *Bad Moon Rising* [19] [21] .
7. **Advanced (10–15):** Numerous misinformation sources, multiple kill sources and alignment/character changes. Requires experienced players.
8. **Expert/Chaos (> 15):** Highly chaotic scripts with overlapping misinformers, multi-kill demons, frequent character or alignment swaps, and little protection. Best reserved for experts or experimental games.

Example Evaluation

Example 1 – 8-player Trouble Brewing: Chef, Empath, Fortune Teller, Undertaker, Virgin, Drunk (Outsider), Poisoner (Minion), Imp (Demon) [37] . Tags: Chef (info), Empath (info), Fortune Teller (info), Undertaker (info), Virgin (onceInfo), Drunk (misinfo), Poisoner (misinfo), Imp (kill). Base weight ≈ (−1×4) + (−0.5) + (2) + (2) + (1) = **3.5**. There is only one misinfo tag (Drunk/Poisoner), and one kill source, so synergy adjustments are small. After normalisation the score may fall around **4**, categorising the script as *Beginner*.

Example 2 – 9-player custom script with Vortox, Pit-Hag, Witch, No Dashii, Fang Gu, Dreamer, Barber, Snake Charmer, Mathematician: tags: multiple misinformers (Poisoner via No Dashii, Vortox), character change (Pit-Hag, Snake Charmer), alignment change (Fang Gu), madness (Witch), multi-kill (No Dashii poisons but still one kill), plus information roles (Dreamer, Mathematician). Base weight ≈ + (2+2 misinfo) + (2 charChange) + (3 alignChange) + (2 madness) + (1 kill) − (1 info ×2) ≈ **11**. Additional synergy for multiple misinformers (+1) and both alignment and character change (+1) raises score to ≈13. Normalised for 9 players gives ≈7, which falls into the *Intermediate/Advanced* category.

The algorithm can be refined empirically by comparing scores of known balanced scripts and adjusting weights accordingly. The key is that the scoring is deterministic and transparent so designers can see which elements increase or decrease the difficulty.

Implementation Considerations

- **Metadata Source:** Character weights and tags should be maintained in a configuration file (e.g. YAML/JSON) separate from code. This allows updates as new characters are released or as community consensus on difficulty changes.

- **First-Night and Other-Night Ordering:** For custom scripts, night order can be derived by listing all characters present and sorting by their `nightOrder`. The official script tool provides night sheets; developers may digitise these or allow script authors to specify order manually. The engine should also allow exceptions when abilities trigger out of order [28].
- **User-generated Scripts:** The script manager should check for illegal combinations (e.g. multiple Demons in a non-Teensyville game, invalid number of Outsiders) and enforce modifications from characters like Baron or Drunk. It should then compute difficulty and display a warning if the script is extremely chaotic.
- **Storyteller Intervention:** Although the digital engine automates rules, a human Storyteller (or moderator AI) is often required to judge madness, answer player questions and provide context. The AI module can offer deterministic decisions for events that usually require judgement (e.g. which players a neutral misinforming character targets) but should be configurable.
- **Accessibility and On-boarding:** Since some editions are labelled "Intermediate" or "Beginner" [18] [19], the UI can recommend easier scripts to new players and hide complex options behind an "advanced" toggle.

Conclusion

Designing a digital version of **Blood on the Clocktower** demands careful modelling of characters, states and event order. The engine must enforce secrecy, alternate day and night phases, handle complex abilities and state changes, and allow a human or AI Storyteller to guide the experience. Building a deterministic script difficulty scoring algorithm provides designers with a quantitative tool to balance their custom scripts. By tagging characters, assigning heuristic weights based on official edition descriptions and community insights, and applying synergy adjustments, the algorithm can classify a script as beginner, intermediate, advanced or chaotic. This framework will help ensure that digital games retain the tension, bluffing and logic puzzles that make the tabletop version beloved while making it accessible and fair for remote players.

---

[1] [2] [3] [4] [5] [6] [35] Rules Explanation - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Rules_Explanation

[7] [28] [29] [30] [32] Abilities - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Abilities

[8] [24] [25] [26] [27] [37] Setup - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Setup

[9] [18] Trouble Brewing - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Trouble_Brewing

[10] Vortox - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Vortox

[11] Travellers - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Travellers

[12] Fabled - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Fabled

[13] Deus ex Fiasco - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Deus_ex_Fiasco

[14] [15] [16] [17] [31] [33] [34] States - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/States

[19] [20] Sects & Violets - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Sects_&_Violets

[21] [22] Bad Moon Rising - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Bad_Moon_Rising

[23] Teensyville - Blood on the Clocktower Wiki
https://wiki.bloodontheclocktower.com/Teensyville

[36] Script Building | Ravenswood Bluff
https://ravenswoodbluff.com/script-building/